Medida borrosa de la Fiabilidad de un Recurso para Entornos Grid

Trabajo Fin de Máster en Ingeniería de Computadores

Curso 2010/2011

Máster en Investigación en Informática
Facultad de Informática
Universidad Complutense de Madrid

Germán P. Santos Benavides

Director: Rubén S. Montero
Colaborador: Luis Garmendia

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Medida borrosa de la fiabilidad de un recurso para entornos Grid", realizado durante el curso académico 2010-2011 bajo la dirección de Rubén Santiago Montero en el Departamento de Arquitectura de Computadores y con la colaboración externa de dirección de Luis Garmendia, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Germán P. Santos Benavides

Índice general

Índice de figuras

Índice de cuadros

Índice de listados

Prólogo

Resumen

La planificación de trabajos en Grids Computacionales es una tarea compleja debido a la falta de control sobre los recursos. En estas condiciones es imposible saber exactamente qué recurso se comportará mejor para cada trabajo.

Por otra parte, los sistemas borrosos han demostrado en los últimos años que funcionan perfectamente en estos escenarios en los cuales la información es imprecisa o incompleta y donde se carece de un modelo exacto del espacio de optimización. En esta memoria presentamos un sistema borroso, construido para trabajar con el meta-planificador GridWay, para evaluar la fiabilidad de cada recurso candidato. Esta información puede ser después utilizada por GridWay para elegir el recursos al que el trabajo será lanzado.

La memria está dividida en 3 partes. Las dos primeras son más teóricas y sirven de introducción a las dos tecnologías sobre las que se basa este trabajo, la Lógica Borrosa y la meta-planificación. La tercera está completamente dirigida a explicar las soluciones propuestas por proyecto del que surge esta memoria y su posterior diseño e implementación.

Palabras clave

Superscheduling; Metaplanificador GridWay; Computación Grid; Lógica Borrorsa; Razonamiento Aproximado; Soft computing

Summary

Job scheduling in Computational Grids is a complex task because of the lack of control over the resources. Under these conditions, it is impossible to know exactly which host will perform better for a given job.

On the other hand, fuzzy systems have demonstrated in recent years to behave well in this kind of situations, with imprecision, incomplete information or a lack of an exact model of the optimization space. In this paper we present a fuzzy system, built on top of the GridWay meta-scheduler, to evaluate the reliability of each candidate resource. This information is used by GridWay to choose the resource where the job will be dispatched.

This memo is divided in three parts. The first two are an introduction to the theories on which this work is based upon, the Fuzzy Logic and the superscheduling. The last part is completely dedicated to explain the solutions proposed by this project and its design and implementation.

Keywords

Superscheduling; GridWay meta-scheduler; Grid computing; Fuzzy Logic; Approximate reasoning; Soft computing

Agradecimientos

La presente Memoria del Máster en Investigación en Informática 2011 comenzó como un intento de aglutinar en un mismo proyecto dos tecnologías a las que veo un potencial bastante significativo, los sistemas distribuídos y la Lógica Borrosa.

Me gustaría agradecer a Rubén Santiago Montero y a Luis Garmendia toda la ayuda que me han prestado a lo largo del proyecto. Gracias a esta ayuda fuimos capaces de presentar un artículo para el congreso ISKE de 2009. El proceso de escritura de este artículo, además, aclaró algunos flecos del proyecto considerablemente y permitió encontrar finalmente una estrategia adecuada para introducir la Lógica Borrosa en el metaplanificador GridWay.

Capítulo 1

Introducción

Este capítulo realiza una descripción de las motivaciones y objetivos de la presente memoria. Se estructura en tres secciones: la sección 1.1 que muestra las principales motivaciones que han dado lugar al desarrollo de este proyecto, La sección 1.2 expone los objetivos que se pretenden conseguir e introduce brevemente la estrategia que será desarrollada más adelante en esta memoria. Y finalmente, la sección 1.3 presenta la estructura general de la memoria.

1.1. Motivación

Los fallos, los cortes de servicio y las violaciones de los contratos de rendimiento son fenómenos frecuentes en sistemas grid. Es un entorno muy dinámico y complejo en el que la información disponible sobre los recursos tiene un cierto grado de incertidumbre. En este contexto, emerge la Lógica Borrosa como una solución adecuada para implementar un método de decisión que permita trabajar con esa incertidumbre. Esta memoria presenta una nueva estrategia de selección de recursos para el metaplanificador GridWay basada en la Lógica Borrosa.

GridWay, como *middleware* metaplanificador, es responsable de detectar y replanificar los trabajos que hayan fallado debido a degradaciones de rendimiento o cortes de servicio. Cada una de estas migraciones tiene una penalización en el rendimiento total del sistema. Por tanto, una mejora en el criterio de selección del mejor recurso posible del conjunto de disponibles supondrá reducir la frecuencia de migración y tendrá un impacto positivo en el rendimiento. Para ello se utilizará toda la información disponible de los recursos y será tratada utilizando técnicas de inferencia borrosa para obtener una medida de fiabilidad de cada recurso.

Existen numerosos intentos de optimizar la planificación en sistemas Grid utilizando lógica borrosa. En [17] se describe una forma de aplicar asociación borrosa de reglas para analizar los datos de monitorización y de contabilidad. El objetivo de esta técnica es la extracción de los patrones de utilización de los recursos a partir de esos datos para predecir la evolución del uso de los recursos. En función de esta información el planificador es capaz optimizar las políticas de planificación.

Un acercamiento diferente es peresentado en [14]. En este trabajo se considera la utilización de un planificador borroso por encima del planificador local para optimizar la distribución de la carga de trabajo. Este planificador decide, utilizando técnicas de control borroso, si la ejecución del trabajo debe ser llevada a cabo localmente o si, por el contrario, debe ser delegada a un sistema remoto. El controlador borroso presentado es adaptativo: el peso de cada regla en la decisión se va actualizando en función de las medidas de rendimiento obtenidas.

1.2. Objetivos

El objetivo del proyecto es optimizar el método de selección de recursos utilizado por GridWay. Para ello se construye un sistema de lógica borrosa que permite estimar la fiabilidad de cada recurso de manera más exacta que el sistema actual, basándose en toda la información de monitorización y de contabilidad proporcionada por GridWay. Una vez calculado el valor de la fiabilidad de los recursos candidatos, el trabajo será lanzado al recurso más fiable.

La información de un recurso se analiza por separado. Por un lado, un recurso puede ser fiable desde el punto de vista de monitorización porque parece desocupado y tiene elevada capacidad de proceso. Por otro lado, las anteriores ejecuciones que se han llevado a cabo en el recurso pueden servir como guía sobre el comportamiento de futuros trabajos. Para diferenciar los dos tipos de información será necesario disponer de un conjunto borroso sobre el conjunto de recursos disponibles para modelar cada uno de los conceptos: el conjunto *mon* aglutina toda la información de monitorización mientras que el conjunto *acct* agrupa toda la información de contabilidad.

Los conjuntos *mon* y *acct* son definidos por el usuario para cada trabajo. Para expresar las características deseadas que debería tener el recurso objetivo, el usuario construye las reglas como una combinación de restricciones sobre las variables del modelo abstracto de recurso atendiendo a las necesidades de cada trabajo. Las restricciones se combinan utilizando las operaciones básicas `AND`, `OR`, `NOT`, y son incluidas en el *job template*.

El siguiente ejemplo muestra un ejemplo de configuración que podría ser aplicada a un trabajo de cálculo intensivo:

```
mon = cpu_load IS low AND cpu_speed IS high
acct = error_rate IS low AND execution_time IS high
```

Esto es considerado una mejora sobre la política de rank descrita en la sección 4.2.2, porque el usuario puede definir sus preferencias sobre las características del recurso necesario de una forma que imita a su proceso de razonamiento. Así, el usuario puede centrarse en qué características quiere y no en cómo expresar sus preferencias.

1.3. Estructura de la memoria

La memoria está divida en tres grandes bloques:

- El primero de ellos hace una breve introducción a los conceptos necesarios de Lógica Borrosa que sirven como báse teórica a los capítulos posteriores. Se presenta el concepto de conjunto borroso y se definen las operaciones para construir un álgebra de conjuntos borrsos. También se describen los conceptos de variable lingüística y proposición borrosa.

- El segundo bloque presenta la Computación Grid, explicando toda la problemática singular que supone la planificación en un entorno de este tipo. Se detalla, también, la estrategia que sigue GridWay para hacer frente al dinamismo del Grid y se introducen los conceptos de planificación y ejecución adaptativas.

- El tercer bloque hace una descripción pormenorizada de las partes específicas a este proyecto: el cálculo de la medida borrosa de fiabilidad, su implementación y los resultados obtenidos. Específicamente, el capítulo 5 presenta la estrategia y su implementación, mientras que el capítulo 6 muestra la implementación del banco de pruebas con la librería de simulación GridSim y analiza los resultados obtenidos.

Al final se presenta un capítulo con las conclusiones obtenidas a partir de este proyecto y qué perspectivas abre de cara al futuro.

Parte I

Introducción a la Lógica Borrosa

Capítulo 2

Conjuntos borrosos

El crecimiento en la complejidad de los problemas que se intentan resolver provoca que obtener un modelo exacto de esos problemas puede ser una tarea compleja o, incluso, inabarcable. Lofti A. Zadeh explica esa situación en [33][34] usando el *principio de incompatibilidad* el cual enucia de la siguiente forma:

> *"A medida que la complejidad de un sistema crece nuestra habilidad para hacer afirmaciones precisas o, incluso, relevantes sobre su funcionamiento desciende. Puede llegarse a un punto límite en el que la precisión y la relevancia sean características mutuamente excluyentes."*

Por otra parte, el razonamiento humano es capaz de tomar decisiones racionales en un entorno impreciso. Por ejemplo, mientras conducimos un coche no se piensa el número de grados exactos que hay mover el volante, el punto exacto donde debemos empezar a frenar o la fuerza que debemos aplicar al pedal de freno. La idea de la Lógica Borrosa es aprovechar esta toleracia a la imprecisión que tienen algunos sistemas[35], permitiendo el manejo de información imprecisa, inexacta o subjetiva de la misma forma que puede hacerlo el pensamiento humano.

En esta parte se presentan los conceptos básicos de la Lógica Borrosa que serán utilizados a lo largo de la memoria. Este paradigma permite modelar conceptos como *alto* o *viejo* y razonar con ellos imitando la forma en que lo hace el razonamiento humano, aunque sean conceptos sujetos a imprecisión, inexactitud, ruido o subjetividad, siendo capaz de obtener conclusiones válidas.

Por último, cabe señalar que, aunque el contenido de esta parte intenta ser autocontenido, el campo de la Lógica Borrosa es realmente amplio y no se pretende tratarlo con profundidad en esta memoria. Los conceptos fundamentales serán definidos de forma clara para formar una base suficientemente amplia para comprender el uso que se hace de estas técnicas en el proyecto. El lector interesado puede remitirse a los artículos [32, 34, 33] donde se explican y se desarrollan todos los aspectos en mayor profundidad.

2.1. Introducción

Un conjunto clásico A, según la teoría clásica de Cantor, se define de tal forma que para cualquier elemento $x \in U$, donde U es el universo de discurso, debe cumplirse que $x \in A$ o $x \notin A$. Esta afirmación ya no es cierta con en la Lógica Borrosa ya que, para representar matemáticamente conceptos como *templado* o *alto* que son inexactos y subjetivos, los conjuntos borrosos permiten definir un *grado de pertenencia* de un elemento a un conjunto.

Definición 1 (Conjunto borroso) *Un conjunto borroso A sobre un universo U se define mediante una* función de pertenencia $\mu_A : U \to [0, 1]$. *La expresión $\mu_A(x)$ es el* grado de pertenencia *de x al conjunto borroso A. Este grado de pertenencia es interpretado normalmente como el valor de un término de una variable lingüística [30, 34], o para medir una característica aplicada a todos los elementos de un universo U.*

Los conjuntos borrosos suponen, por tanto, una generalización del concepto de conjunto, ya que los conjuntos clásicos también pueden ser definidos como un conjunto borros con la siguiente función característica:

$$\mu_B : U \to \{0, 1\}$$
$$\mu_B(x) = \begin{cases} 1 & \text{if } x \in B \\ 0 & \text{if } x \notin B \end{cases}$$

En el ámbito de la Lógica Borrosa, se utiliza el término *crisp* (nítido) para denominar a los conjuntos clásicos. Puede observarse la diferencia existente entre ambas funciones en la figura 2.1.

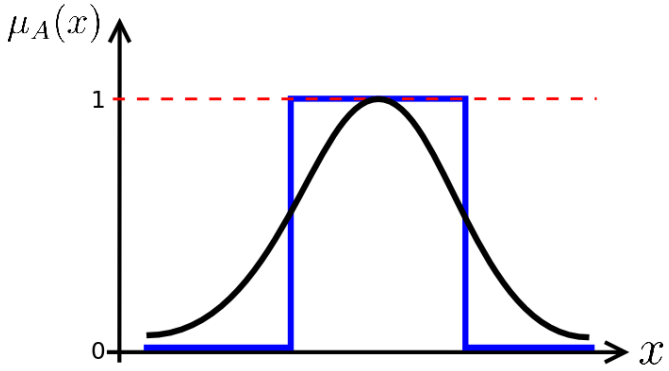


Figura 2.1: Comparación entre un conjunto borroso (negro) y otro *crisp* (azul)

2.2. Operaciones con conjuntos borrosos

Una vez modeladas las características A y B con los conjuntos borrosos $\mu_A, \mu_B : U \to [0, 1]$ puede construirse un álgebra de conjuntos borrosos definiendo las operaciones de

intersección, unión y complemento de conjuntos borrosos. Para ello se utilizarán unos operadores llamados normas triangulares (*t-normas*), conormas triangulares (*t-conorma*) y operadores de negación [27, 21].

2.2.1. Intersección de conjuntos borrosos

En la teoría clásica al definir un conjunto como la intersección de otros dos se obtiene un conjunto con aquellos elementos que pertenecen a ambos conjuntos a la vez. Intuitivamente, al intersecar dos conjuntos borrosos A y B, debe obtenerse un conjunto $A \cap B$ cuya función característica $\mu_{A\cap B}(x)$ sea capaz de expresar el grado de pertenencia de cada elemento $x \in U$ a ambos conjuntos combinando los valores de $\mu_A(x)$ y $\mu_B(x)$. En la Lógica Borrosa esta función de pertenencia $\mu_{A\cap B}$ es definida utilizando una t-norma T, de tal forma que:

$$\mu_{A\cap B} = T(\mu_A(x), \mu_B(x)) \text{ para todo } x \in U$$

No existe un único operador binario que defina la intersección de conjuntos borrosos. Hay múltiples operadores que pueden ser utilizados, siempre que cumplan con las condiciones enunciadas en la siguiente definición:

Definición 2 (T-norma[27]) *Una operación binaria $T : [0,1] \times [0,1] \to [0,1]$ es una t-norma[22, 30], si satisface los siguientes axiomas:*

***Condición de contorno**: $T(1, x) = x$*
***Simetría**: $T(x, y) = T(y, x)$*
***Asociatividad**: $T(x, T(y, z)) = T(T(x, y), z)$*
***Monotonía**: $x \leq x', y \leq y' \Rightarrow T(x, y) \leq T(x', y')$*

Como ejemplos básicos de t-normas se muestran el mínimo, propuesta por Zadeh, y el producto:

mínimo	$T_{min}(x, y) = \min(x, y)$
producto	$T_{prod}(x, y) = xy$

2.2.2. Unión de conjuntos borrosos

De manera análoga a la intersección, dados dos conjuntos borrosos A y B con funciones características μ_A y μ_B respectivamente debe definirse un operador capaz de expresar el conjunto unión $A \cup B$ combinando los valores de $\mu_A(x)$ y $\mu_B(x)$. La unión de conjuntos borrosos se realiza en la Lógica Borrosa con los operadores llamados t-conormas. Por tanto, la función característica que define el conjunto unión $A \cup B$ se expresa de la siguiente manera:

$$\mu_{A\cup B} = S(\mu_A(x), \mu_B(x)) \text{ para todo } x \in U$$

Un operador binario debe cumplir los axiomas expuestos en la definición 3 para ser considerado una conorma.

Definición 3 (T-conorma[27]) *Una operación binaria $S : [0,1] \times [0,1] \to [0,1]$ es una t-conorma[22, 30], si satisface los siguientes axiomas:*

***Condición de contorno**: $S(0,x) = x$*
***Simetría**: $S(x,y) = S(y,x)$*
***Asociatividad**: $S(x,T(y,z)) = S(T(x,y),z)$*
***Monotonía**: $x \leq x', y \leq y' \Rightarrow S(x,y) \leq S(x',y')$*

Los ejemplos básicos de t-conormas son el máximo, propuesta por Zadeh, y la suma probabilística:

máximo	$S_{min}(x,y) = \max(x,y)$
suma probabilística	$S_{prod}(x,y) = x + y - xy$

2.2.3. Complemento de un conjunto borroso

El complemento de un conjunto borroso A se define aplicando un operador negador $N(x)$ a su función característica μ_A obteniendo la función $\mu_{\bar{A}}(x) = N(\mu_A(x))$ para todo $x \in U$. El operador $N(x)$ debe cumplir los axiomas presentados en la siguiente definición.

Definición 4 (Complemento) *Un operador unario $N : [0,1] \to [0,1]$ es un complemento si satisface los siguientes axiomas:*

$$N(0) = 1$$
$$N(1) = 0$$
$$x \leq x' \Rightarrow N(x) \geq N(x')$$

*Una negación es **estricta** si es continua y estrictamente decreciente. Una negación es **involutiva** si $N(N(x)) = x$ para todo $x \in [0,1]$. Una negación es **fuerte** si es estricta e involutiva.*

Como ejemplos de negaciones fuertes se presentan el operador estándar de complemento y las negaciones de Yager:

estándar	$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$
Yager	$\mu_{\bar{A}}(x) = (1 - \mu_A(x)^\alpha)^{\frac{1}{\alpha}}$ siendo $\alpha > 0$

2.2.4. Dualidad de los operadores

A partir de una t-norma T y una negación N se puede definir una t-conorma dual S_T como:

$$S_T(x,y) = N(T(N(x),N(y)))$$

Como puede observase, esta definición supone una generalización de las leyes de De Morgan.

2.2.5. Familias lógicas

Las operaciones algebraicas de intersección, unión y complemento, descritas a lo largo de esta sección, se utilizan normalmente para dotar de sentido a los operadores lógicos *and*, *or* y *not* respectivamente. Por tanto, una familia de conectivos lógicos borrosos (T, S, N) está formada por una t-norma T, una t-conorma S y una negación N. Esta familia de conectivos se denomina **terna de De Morgan** cuando S es la t-conorma dual de T respecto a la negación N.

Las familias que van a ser utilizadas en este proyecto son la familia estándar o de Zadeh (ver imagen 2.3), la familia algebraica o del producto (ver imagen 2.2) y la familia de Lukasiewicz (ver imagen 2.4).

2.2.6. Otros tipos de agregación

Hay conceptos que no se pueden modelar totalmente con los operadores de intersección, unión y negación. Cuando se intentan conjugar diferentes criterios que deberían tener mayor o menor relevancia en la decisión final se utilizan operadores de agregación. Esta clase de operadores se encuentran entre las t-normas y las t-conormas, cumpliéndose la siguiente desigualdad:

$$\text{t-norma } \leq min \leq \text{agregación } \leq max \leq \text{t-conorma}$$

Los operadores de agregación que han sido utilizados durante el desarrollo del sistema borroso utilizan un vector de pesos, aplicando un factor de relevancia a cada uno de los conjuntos que se pretenden agregar. Un vector $w = (w_1, \ldots, w_n)$ es un vector de pesos si $w_i \in [0, 1]$ y $\sum_{i=1}^{n} w_i = 1$.

Definición 5 (Agregación Media Ponderada[10]) *Dado un vector de pesos $w = (w_1, \ldots, w_n)$ y un conjunto de conjuntos borrosos $\{\mu_1, \ldots, \mu_n\}$ la media ponderada se define como*

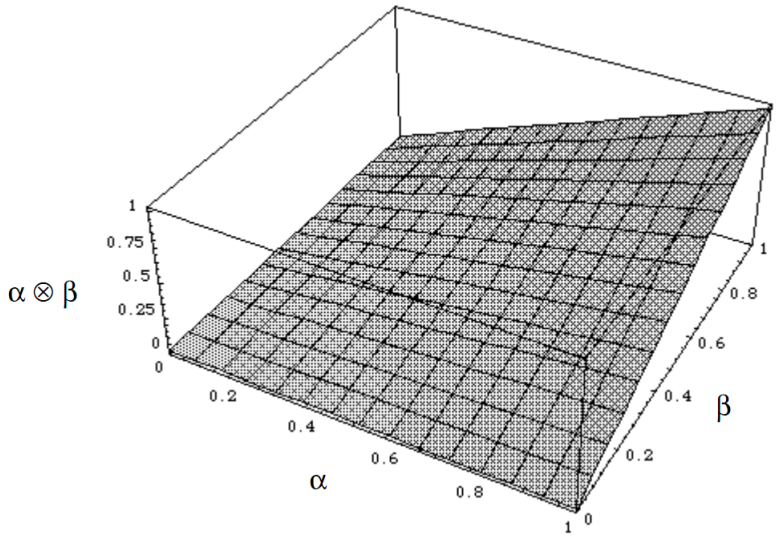
$$\mu_{WA}(x) = \sum_{i=1}^{n} w_i \mu_i(x) \tag{2.1}$$

Definición 6 (Agregación Media Ponderada Ordenada[29, 10]) *Dado un vector de pesos $w = (w_1, \ldots, w_n)$ y un conjunto de conjuntos borrosos $\{\mu_1, \ldots, \mu_n\}$ la media ponderada ordenada se define como*

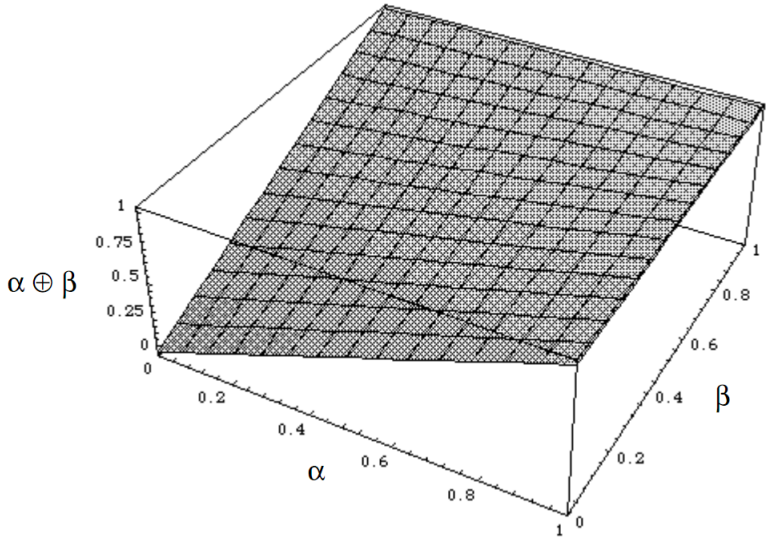
$$\mu_{OWA}(x) = \sum_{i=1}^{n} w_i \mu_{(i)}(x) \tag{2.2}$$

donde $\mu_{(i)}$ representa el valor i-ésimo en la serie ordenada $\mu_{(1)} \leq \mu_{(2)} \leq \ldots \leq \mu_{(n)}$

$$\text{Producto} \begin{cases} T_{prod}(x,y) &= xy \\ S_{prod}(x,y) &= x+y-xy \\ N_{prod}(x) &= 1-x \end{cases}$$



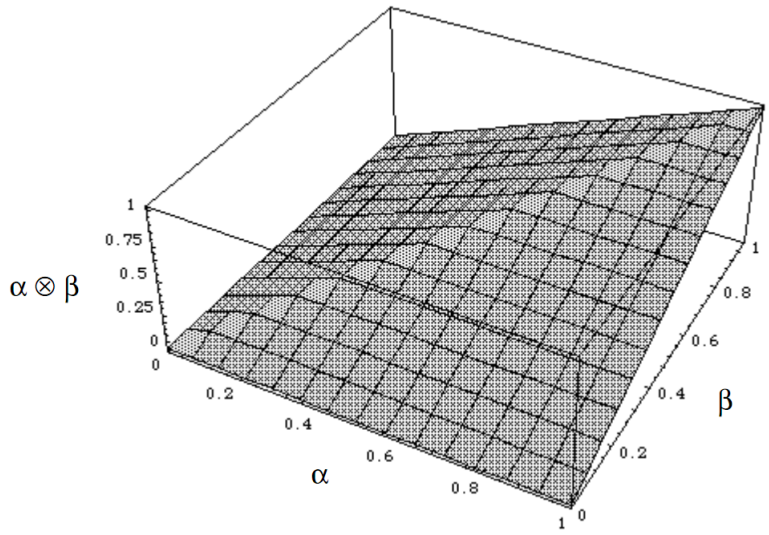
(a) T-norma del producto



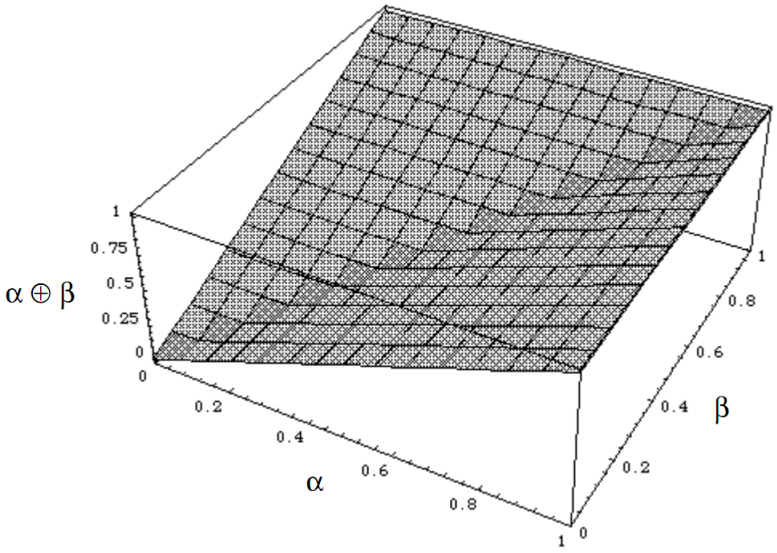
(b) T-conorma del producto

Figura 2.2: Familia lógica algebraica o del producto

$$\text{Zadeh} \begin{cases} T_{zadeh}(x,y) &= min(x,y) \\ S_{zadeh}(x,y) &= max(x,y) \\ N_{zadeh}(x) &= 1-x \end{cases}$$



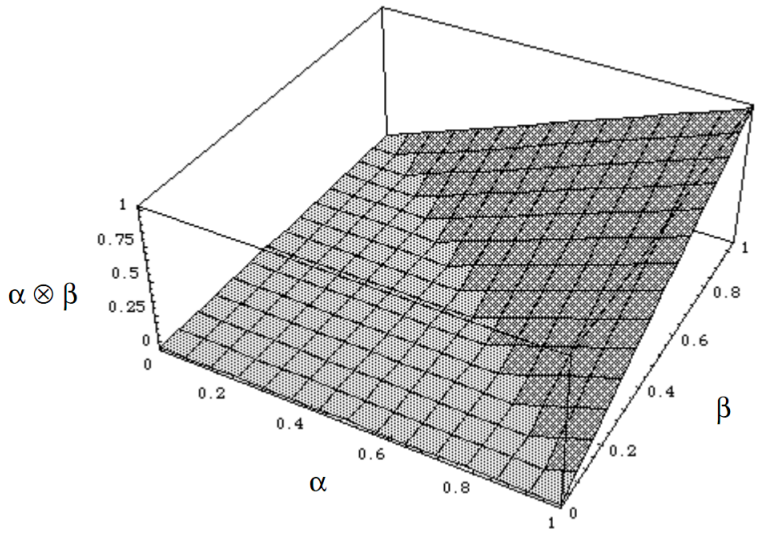
(a) T-norma de Zadeh



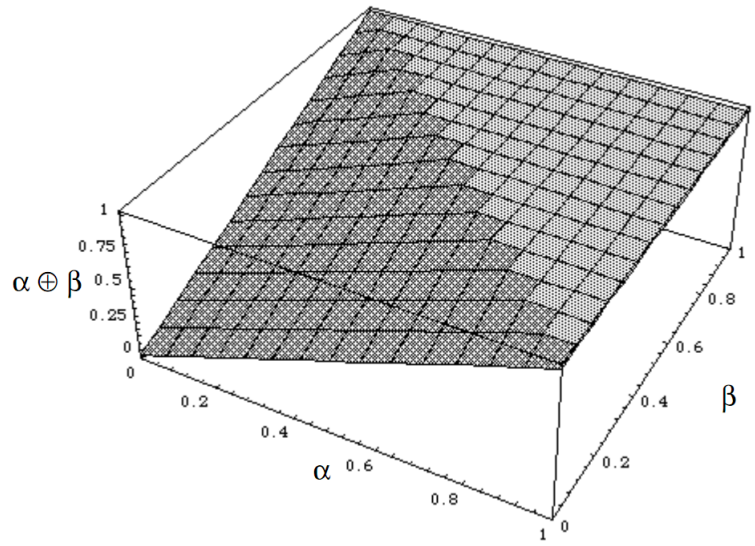
(b) T-conorma de Zadeh

Figura 2.3: Familia lógica estándar o de Zadeh

$$\text{Lukasiewicz} \begin{cases} T_{luk}(x,y) &= max(0, x+y-1) \\ S_{luk}(x,y) &= min(1, x+y) \\ N_{luk}(x) &= 1-x \end{cases}$$



(a) T-norma de Lukasiewicz

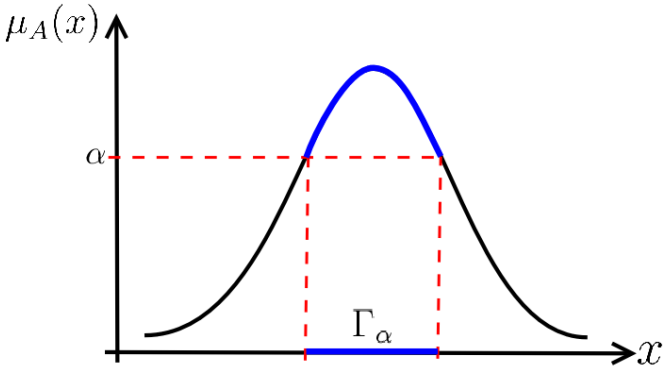


(b) T-conorma de Lukasiewicz

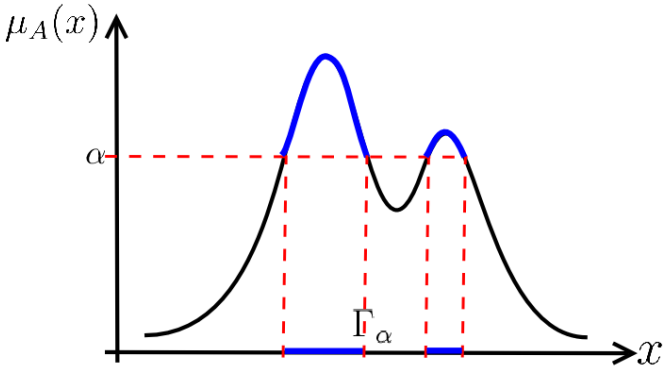
Figura 2.4: Familia de Lukasiewicz

2.3. Convexidad en los conjuntos borrosos

La convexidad de los conjuntos borrosos tiene relevancia en los sistemas de control borroso puesto que es más fácil, computacionalmente hablando, trabajar con ese tipo de conjuntos[24]. La definición de convexidad se basa en los llamados *corte-α* de un conjunto borroso A o Γ_α. El conjunto Γ_α es un conjunto clásico, formado por todos los elementos del dominio cuyo valor de la función característica sea igual o mayor que $\alpha \in (0, 1]$.



(a) Conjunto borroso convexo.



(b) Conjunto borroso no convexo.

Figura 2.5: Convexidad de un conjunto borroso

Definición 7 (Convexidad[32]) *Sea A un conjunto borroso sobre un universo U definido por su función característica μ_A y sea $\alpha \in (0, 1]$. A será convexo si y sólo si todos los conjuntos*

$$\Gamma_\alpha = \{x | \mu_A(x) \geq \alpha\}$$

son convexos.

Como recordatorio, se dice que un conjunto clásico C es convexo si dados dos puntos $x_1, x_2 \in C$ se cumple que todos los elementos del universo que se encuentren entre x_1 y x_2 pertenecen también al conjunto, o expresado matemáticamente $\theta x_1 + (1 - \theta)x_2 \in C$, con $\theta \in [0, 1]$. Se puede comparar gráficamente los diferentes tipos de función característica que tienen un conjunto borroso convexo y no convexo en la figura 2.5.

Capítulo 3

Variables lingüísticas

3.1. Introducción

La forma tradicional de modelar un sistema es cuantitativamente, atendiendo a medidas exactas y manipulando números y símbolos. El pensamiento humano, por otra parte, utiliza en su mayor parte palabras del lenguaje natural y percepciones en su proceso de razonamiento. Siguiendo ese modelo, en Lógica Borrosa conceptos como *altura* o *temperatura* pasan a tener valores que no son más que etiquetas lingüísticas. En el ejemplo de la figura 3.1 se encuentra representada la variable *Temperature* con el dominio de términos $\{cold,\ cool,\ confortable,\ warm,\ hot\}$.

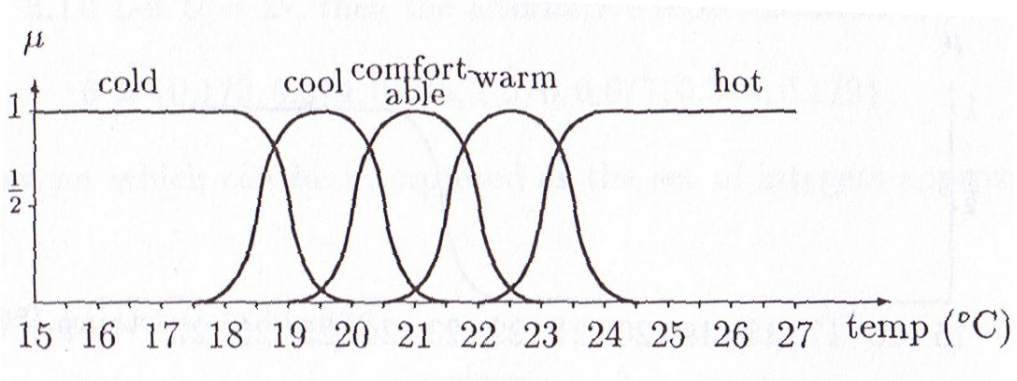


Figura 3.1: Variable lingüística *Temperature*

Zadeh define en [34][31] una variable lingüística como una variable cuyo dominio no son números sino palabras o frases en un lenguaje natural o artificial. A los valores del dominio se les conoce como *términos* y tienen asociado un conjunto borroso que les dota de significado. La función de pertenencia de estos términos normalmente está construída sobre una la *variable base* de naturaleza numérica. En el ejemplo de la figura 3.1 la variable base es $temp \subseteq \Re$.

En los sistemas de Control Borroso la forma que se elije para los conjuntos borrosos que definen a los términos de una variable lingüística suele ser triangular o trapezoidal,

debido a su eficiencia computacional y de almacenaje [13]. Esto es así porque este tipo de funciones pueden ser representadas con 3 o 4 puntos respectivamente. Es conveniente, además, que dada una variable lingüística A con el dominio de términos $\{A_1, \dots A_n\}$ que las funciones de pertenecia de estos conjuntos borrosos cumplan una serie de propiedades, que pueden ser observadas en la figura 3.2. Las propiedades más importantes son:

Normalidad: debe existir un elemento $x \in U$ tal que $\mu_{A_i}(x) = 1$.

Completitud: la función de pertenencia $\mu_{A_i}(x)$ debe estar definida para todo $x \in U$.

Relevancia: cada conjunto debe ser de un tamaño suficientemente grande como para tratar con el ruido.

Solapamiento adecuado: los términos deben tener un solapamiento adecuado, para evidenciar la *borrosidad* de los conceptos asociados. En la figura puede verse las situaciones a evitar: el *low-coverage*, o conjuntos con un soloapamiento mínimo, y el *high-overlapping*, conjuntos con demasiado solapamiento que son prácticamente indistinguibles.

Convexidad: los términos deben ser convexos.

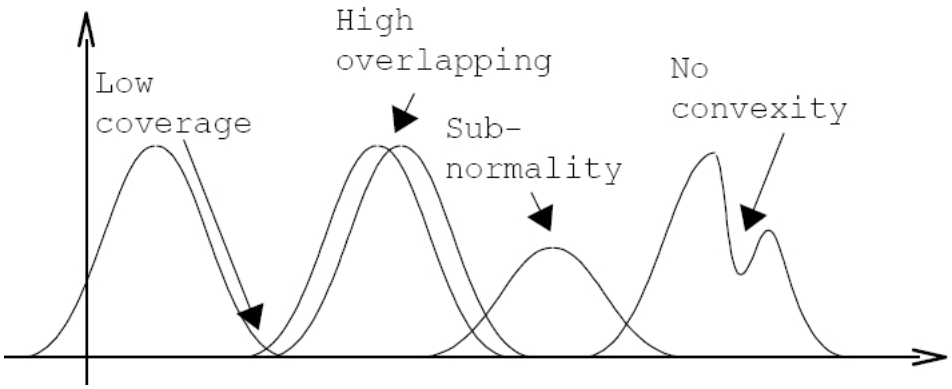


Figura 3.2: Propiedades no deseables de los términos

3.2. Modificadores lingüísticos

Un modificador lingüístico es equivalente a un adverbio en lenguaje natural. Estos operadores son utilizados cuando se pretende concentrar o dilatar la definición de un conjunto borroso. Dada una característica A con su función de pertenencia μ_A podemos modelar el modificador "muy" (*very*) para intensificar:

$$\mu_{very\ A}(x) = (\mu_A(x))^2$$

y el modificador "un poco" (*fairly*) para dilatar:

$$\mu_{fairly\ A}(x) = (\mu_A(x))^{\frac{1}{2}}$$

Estos modificadores permiten definir nuevos términos a partir de los fundamentales del dominio de la variable lingüística. Por ejemplo, dada la variable *Temperature* con el dominio de términos $\{cold,\ cool,\ confortable,\ warm,\ hot\}$ es posible definir los nuevo términos *very hot* o *fairly hot*. La figura 3.3 extraída de [24] presenta gráficamente la variación que suponen los modificadores lingüísticos.

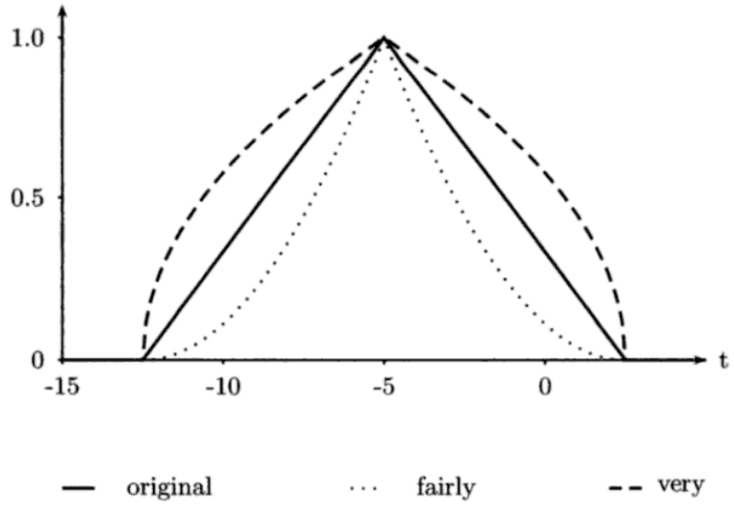


Figura 3.3: Representación gráfica de los modificadores lingüísticos

3.3. Proposiciones borrosas

Una proposición borrosa es una proposición que incluye variables lingüísticas. Una proposición simple es una restricción de la siguiente forma:

$$x \text{ is } T$$

donde x es una variable lingüística y T uno de sus términos. El valor de verdad asignado a una proposición básica viene dado por la función de pertenencia μ_T. Hay que tener en cuenta que el término T puede ser tanto un término básico como generado, ya sea mediante los modificadores *very* y *fairly* o mediante el complemento *not*.

A partir de éstas proposiciones básicas se pueden construir proposiciones compuestas gracias a las conectivas lógicas *and* y *or*. La evaluación del valor de verdad asociado a una proposición compuesta se hace aplicando la t-norma y la t-conorma pertenecientes a la familia lógica elegida.

Como ejemplo final de esta parte de introducción a la Lógica Borrosa y enlazando con toda la teoría sobre lógica borrosa que ha sido presentada anteriormente, se muestra

un ejemplo de evaluación de proposición compleja. La proposición de este ejemplo quiere definir la fiabilidad de un recurso atendiendo a varios de sus aspectos como la velocidad de procesador, la tasa de fallos y la tasa de ejecuciones terminadas con éxito. La idea es definir la función pertenencia al conjunto borroso de los recursos fiables aplicando restricciones a los diferentes aspectos de los recursos:

$$p = \text{cpu\_speed } \textbf{is} \text{ high } \textbf{and } (\text{failure\_rate } \textbf{is} \text{ very low } \textbf{or } \text{success\_rate } \textbf{is} \text{ high})$$
$$p(x, y, z) = min(\mu_{cpu\_high}(x), max(\mu_{failure\_low}(y)^2, \mu_{success\_high}(z)))$$

siendo x, y, z los valores dados a los aspectos del recurso: velocidad de procesador, tasa de fallos y tasa de ejecuciones exitosas respectivamente.

Parte II

Introducción a la Computación Grid

Capítulo 4

Computación Grid

4.1. Introducción

El objetivo de la Computación Grid es proporcionar el acceso a recursos computacionales avanzados de manera robusta mediante un interfaz uniforme, desde cualquier lugar y con un coste mínimo. Esta tecnología ha sido comparada con la red eléctrica[8] por lo que, considerando esta analogía, se puede decir que el objetivo final de la computación grid es que los usuarios puedan disponer de recursos de computación como disponen actualmente de energía eléctrica: los usuarios obtienen electricidad a través de los enchufes sin preocuparse de dónde viene la energía o cómo está siendo generada. Una infraestructura similar permitiría poner las capacidades de la supercomputación en las manos de todo el que lo necesitase en el momento en el que lo necesitase.

La computación Grid, sin embargo, todavía no ha alcanzado ese nivel deseado de difusión. Hoy en día, se encuentra en un punto intermedio, que consiste en llevar la computación distribuída a un nivel superior y permitir el uso compartido de recursos y aplicaciones entre organizaciones.

Existen numerosas definiciones propuestas para definir lo que es un grid. En [8] es definido un grid como una gran colección de recursos heterogéneos, distribuidos geográficamente y pertenecientes a diferentes organizaciones virtuales, que proporciona servicios de computación, compartidos coordinadamente, sin que los usuarios conozcan los recursos involucrados. Otra de las definiciones más extendidas por la comunidad Grid es la propuesta, también por Ian Foster, en [16]:

> *"A Grid is a system that...*
>
> 1. *... coodinates resources that are not subject to a centralized control...*
> 2. *... using standard, open, general-purpose protocols and interfaces...*
> 3. *... to deliver nontrivial qualities of services."*

Esta definición impone tres requisitos fundamentales que todo sistema debe cumplir para que pueda ser considerado como sistema Grid:

1. Debe carecer de un control centralizado, de esta manera se imposibilita la formación de cuellos de botella y además facilita la incorporación de nuevos nodos permitiendo así un mejor aprovechamiento de recursos heterogéneos.

2. Debe estar basado en protocolos e interfaces estándar, abiertos y de propósito general. Así, se simplifica la ejecución de aplicaciones a través del Grid, permitiendo que puedan coexistir aplicaciones con distintas políticas de autenticación, autorización o acceso a los recursos.

3. Debe proporcionar calidades de servicio no triviales. Es decir, la tecnología Grid debe ofrecer calidades de servicio superiores a las proporcionadas por la unión de los nodos individuales.

4.1.1. Estándares y tecnologías actuales

En su intento por avanzar en la globalización de la computación grid, a partir de las ideas propuestas en [15], el Global Grid Forum[1] desarrolló en febrero de 2002 la primera especificación OGSA (Open Grid Servides Architecture). El objetivo era crear una arquitectura orientada a servicios utilizando protocolos estándares que fueran abiertos y de propósito general para actividades comunes en la Computación Grid como autenticación, autorización, descubrimiento y acceso a recursos.

La arquitectura OGSA define a partir de los Web Services, que son independientes de plataforma y utilizan tecnologías con tanta difusión como XML y HTTP, los Grid Services, que son servicios web con extensiones para cumplir con las necesidades del Grid. Las diferencias más relevantes con respecto a los Web Services existentes son:

- La vida de un Grid Service puede ser transitoria ya que, en un sistema distribuído a gran escala, los recursos pueden dejar de estar disponibles. Es necesario manejar su ciclo de vida, su creación y su destrucción.

- Un Grid Service tiene una serie de datos y atributos asociados que definen su estado interno.

- Los Grid Services proporcionan métodos de notificación para que los clientes puedan reaccionar ante los cambios de estado del servicio.

La especificación técnica que definía la forma de implementar la arquitectura OGSA mediante Web Services se llamó OGSI (Open Grid Services Infraestructure)[5]. Esta especificación utilizaba extensiones no estándar del WSDL (Web Services Description Language) y de XML Schema. Esta especificación, sin embargo, no fue bien recibida en la comunidad de los Web Services[12] que argumentó que: (i)no era compatible con los web services existentes ni con las herramientas de desarrollo; (ii)era demasiado densa, estando todo definido en una única especificación; (iii) era demasiado orientada a objetos.

En junio de 2004, en un esfuerzo por coordinarse con la comunidad de Web Services, se desarrolló el estándar WSRF (Web Services Resource Framework). Esta especificación es una refactorización y evolución de OGSI que hace mejor uso de todos los estándares XML

existentes. WSRF está constituido por un conjunto de especificaciones que definen la gestión de los WS-Resources: WS-Resource, WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults y WS-ServiceGroup. WSRF también hace uso de WS-Notification para ofrecer una funcionalidad que OGSI implementaba explícitamente, lo que supone una reducción en complejidad.

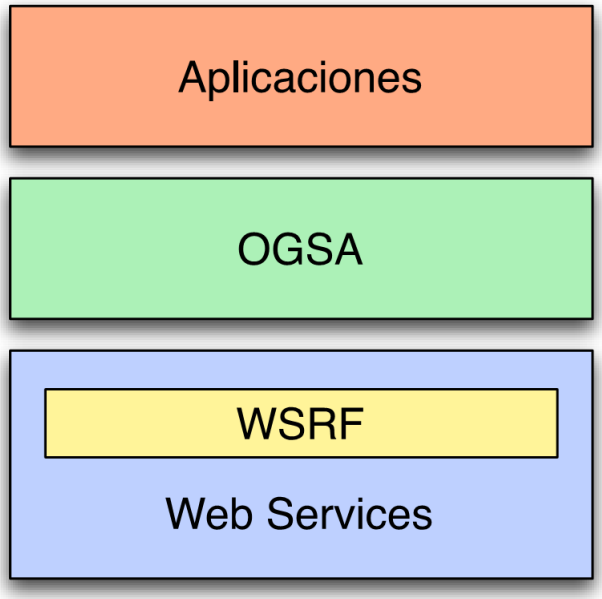


Figura 4.1: Relación entre OGSA, Web Services y WSRF

La figura 4.1 muestra la relación entre la arquitectura OGSA, los Web Services y el estándar WSRF. Por último, hay que señalar que actualmente WSRF v1.2 es un estándar OASIS[7].

4.1.2. Globus Toolkit

La implementación de referencia del estándar WSRF se llevó a cabo con el Globus Toolkit. Este conjunto de herramientas es el estándar *de facto* para la implementación de aplicaciones en sistemas Grid. Globus Toolkit es un software libre, de código abierto y organizado como una colección de componentes debilmente acoplados, cuyo desarrollo está liderado por diversos grupos de investigación ubicados en la *University of Southern California* y en la *University of Chicago*.

4.1.3. Futuro

La computación Grid ha demostrado durante la última década su utilidad para resolver problemas de gran tamaño de ramas tan diversas como la Física, la Bioinformática o las simulaciones climáticas. Sin embargo, ninguna solución comercial ha aparecido para aprovechar esta tecnología en entornos que no sean académicos, limitando así su expansión y desarrollo. Esto es normalmente achacado a la complejidad inherente tanto en el

despliegue como en la gestión de un grid, dado el gran número de recursos y organizaciones involucrados[11].

Por otra parte, una rama relativamente nueva de la computación distribuída como es la Cloud Computing sí cuenta con aplicaciones comerciales con las que potenciar su desarrollo, al centrarse más en ofrecer recursos virtuales bajo demanda o IaaS (Infrastructure-as-a-Service). En colaboración con la tecnología de virtualización, el Cloud Computing permite desplegar una infraestructura de recursos y pagar únicamente por la capacidad utilizada. Este tipo de tecnología permite hacer frente a las fluctuaciones de la carga de trabajo de manera inmediata y, además, reduce el coste asociado. Un ejemplo de aplicación comercial es Amazon EC2, que despliega máquinas virtuales a disposición de los clientes en servidores bajo control del proveedor.

Existen diversos trabajos en curso intentando mejorar la Computación Grid utilizando estas nuevas tecnologías de virtualización y Cloud Computing para simplificar el escenario. Las vías de desarrollo más importantes propuestas en [11] son:

- El uso de técnicas de virtualización y de Cloud Computing como un método para proporcionar a los usuarios del Grid la posibilidad de ejecutar sus aplicaciones en un entorno personalizado. De esta forma, la Computación Grid podría beneficiarse de más flexibilidad, fiabilidad y eficiencia.

- Ofrecer la posibilidad de acceder a los recursos del Grid imitando la manera que pueden accederse en Cloud Computing. Es decir, los usuarios accederían directamente a la capacidad de cálculo, saltándose la capa de *middleware*. Este tipo de acercamiento podría ser una manera natural de atraer inversores hacia las infraestructuras Grid actuales.

4.2. Planificación de recursos en el Grid

La tecnología Grid permite interconectar recursos esparcidos por diferentes dominios administrativos, cada uno con su seguridad y con su sistema de gestión de recursos. El control, por tanto, no puede ser centralizado sino distribuido, lo que impide que los planificadores tengan completo conocimiento del estado del sistema y de las peticiones de los usuarios[16].

La gran dificultad de la planificación en entornos Grid es, por tanto, que se deben tomar decisiones acerca de recursos sobre los que no se tiene control total [26]. No se puede confiar en que una vez enviado satisfactoriamente un trabajo a un recurso, éste vaya a conseguir ejecutarse en él completamente. El trabajo puede ser cancelado o suspendido sin previo aviso, las comunicaciones pueden interrumpirse o el sistema remoto puede fallar. Bajo estas codiciones un metaplanificador tiene que hacer uso de técnicas específicas para obtener el mayor rendimiento posible de un conjunto de recursos y debe contar con características consistentes en detección y recuperación de fallos.

La arquitectura de un planificador de recursos para el Grid debe ser jerárquica [9]. Deben existir:

- planificadores locales (PBS, SGE, Condor, ...) que manejan las políticas dentro de cada nodo.
- *metaplanificadores* (GridWay) que distribuyen los trabajos por los nodos.

En la figura 4.2 puede observarse gráficamente las diferentes capas que componen la arquitectura del metaplanificador GridWay. Las aplicaciones del usuario hacen peticiones al metaplanificador para que planifique un trabajo. Para ello le proporcionan, además del trabajo, una plantilla de trabajo (o *job template*) con sus requisitos. A partir de ese momento es el metaplanificador el que se encarga de la gestión con el Grid de todos los pasos necesarios para conseguir la ejecución del trabajo.

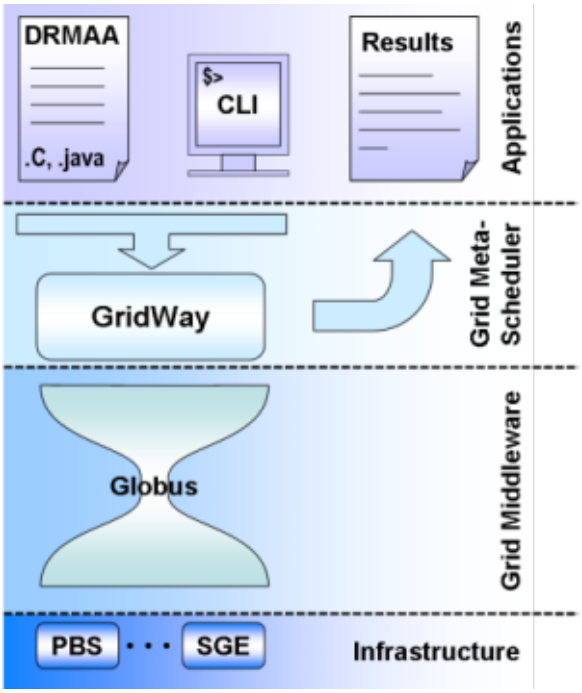


Figura 4.2: Arquitectura del metaplanificador GridWay

El metaplanificador pasa por una serie de fases durante la ejecución de un trabajo[26]:

1. **Descubrimiento de recursos**. Durante esta fase el metaplanificador obtiene un conjunto de recursos activos en los que el usuario tiene permiso para ejecutar trabajos.
2. **Selección del sistema remoto**. Para cada uno de los recursos descubiertos se obtiene su información dinámica. Con esta información y los requisitos del trabajo se evalúan los recursos candidatos y se selecciona el mejor.
3. **Gestión de la ejecución del trabajo**. Una vez seleccionado el recurso, el metaplanificador debe enviar el trabajo y preparar al recurso para la ejecución del trabajo (*staging*). El metaplanificador monitoriza el avance de la ejecución del trabajo y, una vez finalizado, recoge los resultados obtenidos.

4.2.1. Adaptación al medio

Un Grid, como ha sido señalado anteriormente, es un entorno dinámico que complica singularmente la tarea de los metaplanificadores. Para poder sacar el mayor rendimiento posible en esas condiciones, los metaplanificadores deben ser capaces de adaptarse al medio. Las dos estrategias que han sido propuestas para operar en esas condiciones la *planificación adaptativa* y la *ejecución adaptativa*[19][18].

La planificación adaptativa es la primera estrategia que utilizan los metaplanificadores para hacer frente al dinamismo del grid. Consiste en seleccionar el mejor recurso, de todos los disponibles en el Grid que cumplan los requisitos del trabajo, hacia el cual lanzar el siguiente trabajo teniendo en cuenta su estado actual y la información obtenida de ejecuciones de trabajos anteriores.

La ejecución adaptativa, por su parte, es la capcidad de migrar trabajos que ya están corriendo en una máquina hacia un recurso que se ajuste más a sus necesidades basándose en eventos generados dinámicamente tanto por la aplicación como por GridWay. El metaplanificador contempla los siguientes escenarios para realizar una *replanificación dinámica*[20]:

1. Migración iniciada por el Grid
    a) migración oportunista cuando un recurso mejor es descubierto. El metaplanificador debe evaluar si las mejoras en el rendimiento compensarán la migración del trabajo.
    b) pérdida de conexión con el recurso.
    c) el trabajo es cancelado o suspendido.
2. Migración iniciada por la aplicación
    a) violación del contrato de rendimiento. Si el metaplanificador detecta una bajada en el rendimiento inaceptable para el trabajo deberá migrarlo a otro recurso.
    b) petición de la propia aplicación. Una aplicación puede tomar decisiones durante su evolución, modificando sus requisitos y sus expresiones de evaluación de recursos.

4.2.2. GridWay

GridWay es un metaplanificador integrado en el *middleware* Globus Toolkit que permite una ejecución más sencilla (*submit & forget*) y eficiente de los trabajos en entornos grid dinámicos[18]. GridWay realiza de forma automática todas las fases definidas en [26] y descritas anteriormente, referidas como descubrimiento de recursos, selección del sistema remoto y gestión de la ejecución del trabajo.

Planificación en GridWay

Para manejar el dinamismo del Grid, GridWay proporciona mecanismos de recuperación de errores, además de las estrategias de planificación y ejecución adaptativas[19].

El algoritmo de planificación adaptativa de GridWay está descrito en el siguiente extracto de *pseudocódigo* y puede verse gráficamente en 4.3:

for each $job \in Pending \cup Rescheduled$ **do**
  $candidates \leftarrow$ get_candidate_hosts$(job, user, resources)$
  $candidates \leftarrow$ order_by_policy$(candidates)$
lanzar job al recurso get_first_element$(candidates)$

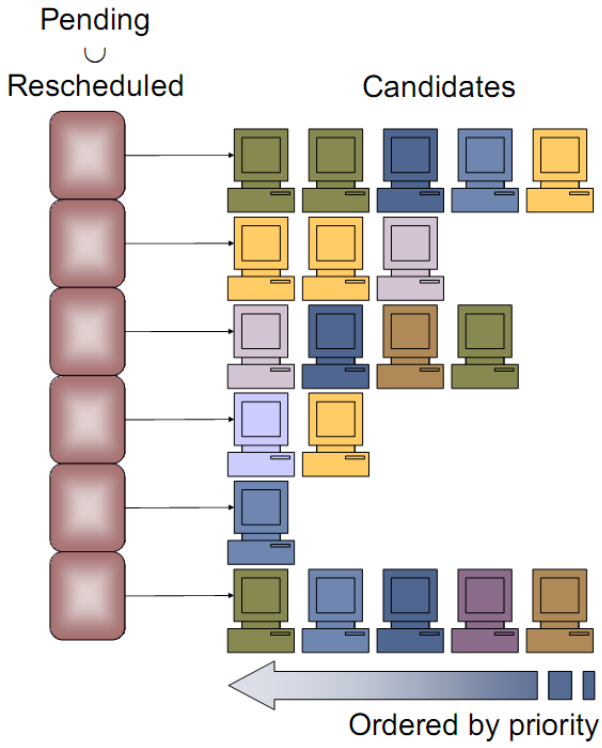


Figura 4.3: Algoritmo de planificación adaptativa de GridWay

Para seleccionar un recurso, el planificador de GridWay primero crea una *meta-cola* con los recursos candidatos, es decir, aquellos que son apropiados para ejecutar ese trabajo para ese usuario atendiendo a las restricciones impuestas por el trabajo (sistema operativo, arquitectura, ...) y a las políticas internas. Después, varias medidas son utilizadas para priorizar los recursos. En particular, GridWay implementa las siguientes políticas:

- **fixed**, para asignar explícitamente una prioridad dada a cada recurso de un Grid.
- **rank**, en la que cada recurso es analizado conforme a una expresión dada por el usuario. Esta expresión evalúa las características como la velocidad de CPU o la cantidad de memoria disponible.

- **usage**, que considera los diferentes tiempos de ejecución de los anteriores trabajos en dicho recurso.

Todas estas políticas son combinadas para obtener el valor de la función de evaluación para ese trabajo y ese usuario. El usuario debe elegir los pesos $w_i \in [0, 1]$ en la ecuación 4.1 para ajustar la importancia de cada una de las políticas individuales:

$$P_{host} = \sum_i w_i p_i \text{ donde } i \in \{\text{fixed, usage, rank}\} \tag{4.1}$$

Parte III

Medida borrosa de la fiabilidad de un recurso

Capítulo 5

Construcción del sistema borroso

Este capítulo presenta toda la información relativa a la concepción del sistema borroso para obtener la medida de fiabilidad de un recurso. El capítulo comienza con una descripción detallada de la estrategia utilizada por el sistema borroso. Una vez presentado el funcionamiento de modo conceptual, en las siguientes secciones se presentan los detalles de implementación del sistema. La sección 5.2 describe la arquitectura a alto nivel, destacando los bloques más importantes y su interacción durante la ejecución de las peticiones de cálculo. Por su parte, la sección 5.3 profundiza en el diseño e implementación de las partes más significativas del planificador, bajando el nivel de abstracción y mostrando las soluciones propuestas a los problemas encontrados durante el desarrollo.

5.1. Estrategia

Este proyecto busca optimizar el método de selección de recursos utilizado por GridWay, descrito en la sección 4.2.2. En un intento de reducir las migraciones y las suspensiones de los trabajos en entornos Grid, se pretende crear un sistema borroso que sea capaz de estimar de forma más aproximada la fiabilidad de un recurso para un trabajo y un usuario concretos.

GridWay tiene información de cada recurso, tanto de su situación actual (información de *host*) como de su comportamiento en anteriores ejecuciones (información de *accounting*). Esta información es utilizada por el sistema borroso para calcular la fiabilidad de cada recurso.

Para aislar, en la medidia de lo posible, al sistema de cálculo borroso de futuros cambios en el *middleware*, toda esta información relativa a cada recurso es condensada en el modelo abstracto de recurso. Los campos del modelo son las siguientes:

- **Información de monitorización**: CPU load, memory load, disk load, nodes available, CPU speed, memory total, disk total, nodes total.
- **Información de contabilidad**: error rate, success rate, suspension rate, transfer time, execution time, suspension time.

El sistema borroso toma como entrada un modelo de recurso para un recurso x, que debe haber sido rellenado previamente con los datos proporcionados por GridWay. La fiabilidad de x es entonces calculada como el grado de pertenencia al conjunto borroso *reliable*, es decir $\mu_{reliable}(x)$.

El conjunto *reliable* se define, a su vez, mediante la agregación de dos conjuntos borrosos que evalúan por separado la información de monitorización, el conjunto *mon*, y la información de contabilidad, el conjunto *acct*. Las funciones de pertenencia de estos conjuntos son definidas por el usuario mediante sendas proposiciones incluídas en el *job template* del trabajo. Estas proposiciones se construyen utilizando las operaciones básicas `AND`, `OR`, `NOT` para combinar restricciones sobre las variables lingüísticas del sistema borroso.

Existe una variable en el sistema borroso para cada uno de los campos del modelo de recurso vistos anteriormente. Las funciones características de los términos de estas variables también serán definidos por el usuario. El listado 5.1 muestra toda la información configurable disponible en el *job template*.

```
FUZZIFY cpu_speed
  TERM high := (0,0) (0.5,0) (1,1);
  TERM medium := (0,0) (0.5,1) (1,0);
  TERM low := (0,1) (0.5,0) (1,0);
END_FUZZIFY

FUZZIFY nodes_total
  TERM high := (0,0) (0.5,0) (1,1);
  TERM medium := (0,0) (0.5,1) (1,0);
  TERM low := (0,1) (0.5,0) (1,0);
END_FUZZIFY

FUZZIFY nodes_available
  TERM high := (0,0) (0.5,0) (1,1);
  TERM medium := (0,0) (0.5,1) (1,0);
  TERM low := (0,1) (0.5,0) (1,0);
END_FUZZIFY

FUZZIFY transfer_time
  TERM high := (0,0) (0.5,0) (1,1);
  TERM medium := (0,0) (0.5,1) (1,0);
  TERM low := (0,1) (0.5,0) (1,0);
END_FUZZIFY

FUZZIFY suspension_time
  TERM high := (0,0) (0.5,0) (1,1);
  TERM medium := (0,0) (0.5,1) (1,0);
  TERM low := (0,1) (0.5,0) (1,0);
END_FUZZIFY
```

```
FUZZIFY success_rate
  TERM high := (0,0) (0.5,0) (1,1);
  TERM medium := (0,0) (0.5,1) (1,0);
  TERM low := (0,1) (0.5,0) (1,0);
END_FUZZIFY

RULES
  FAMILY = MINIMUM;
  MON = (cpu_speed IS high OR cpu_speed IS medium) AND
     nodes_available IS high AND (nodes_total IS high OR
     nodes_total IS medium);
  ACCT = transfer_time IS low AND suspension_time IS low
     AND success_rate IS high;
  REL = 0.67 * ACCT + 0.33 * MON;
END_RULES
```

Listado 5.1: Ejemplo de *job template*

5.1.1. Modelado del conjunto *reliable*

Toda la información necesaria se obtiene a través de GridWay. Sin embargo, mientras la información de contabilidad es mantenida internamente por GridWay, la información de monitorización es obtenida de los *Information Managers* del grid, con un menor nivel de certidumbre. El *Monitoring and Discovery System* (MDS) de Globus, por ejemplo, tiene un modelo de consistencia que permite obtener de él información ligeramente imprecisa u obsoleta[2].

Por tanto, tener varias fuentes de información hace necesario aplicar un criterio de relevancia a la hora de agregar ambos conceptos para definir el conjunto borroso *reliable* sobre el conjunto de los recursos disponibles. Se define el conjunto *reliable* como la media ponderada de ambos conjuntos:

$$\mu_{reliable}(x) = p\ \mu_{mon}(x) + (1-p)\mu_{acct}(x) \text{ donde } p \in [0,1] \tag{5.1}$$

5.1.2. Aplicación a GridWay

Todo lo anterior puede ser aplicado a GridWay modificando el algoritmo de selección de recursos. El algoritmo presentado en la sección 4.2.2 debe ser reescrito para que priorice los recursos según su nivel de pertenencia al conjunto *reliable*:

for each $job \in Pending \cup Rescheduled$ **do**
  $candidates \leftarrow$ get_candidate_hosts$(job, user, resources)$
  for each $host \in candidates$
    $priority \leftarrow p\ \mu_{mon}(host) + (1-p)\mu_{acct}(host)$

$ordered \leftarrow$ insert_ordered($priority, host, ordered$)
dispatch job to get_first_element($ordered$)

5.2. Arquitectura del sistema borroso

En esta sección se presenta la arquitectura de alto nivel del sistema borroso descrito en [25]. La arquitectura del sistema borroso tiene como componentes principales a *FuzzyController*, *DataCollector* y el *FuzzySystem*. En el diagrama de bloques de la figura 5.1 puede se pueden ver las relaciones de los distintos componentes dentro del sistema.

FuzzyController es el bloque responsable de la comunicación con el selector de recursos de GridWay y de sincronizar la ejecución de las diferentes etapas del proceso para cada petición.

DataCollector es el componente que se ocupa de obtener la información necesaria de cada recurso. Dado un recuro y un usuario rellena todas las variables del modelo de recurso que esté definido. Para ello tiene que ser capaz de agregar tanto la información de monitorización como la de contabilidad que puede ser obtenida de GridWay a través de sus comandos `gwhost` y `gwacct`.

FuzzySystem lleva a cabo todo el razonamiento borroso. Recibe toda la información necesaria del recurso y genera la medida de fiabilidad aplicando las definiciones de conjuntos borrosos contenidas en los archivos de configuración.

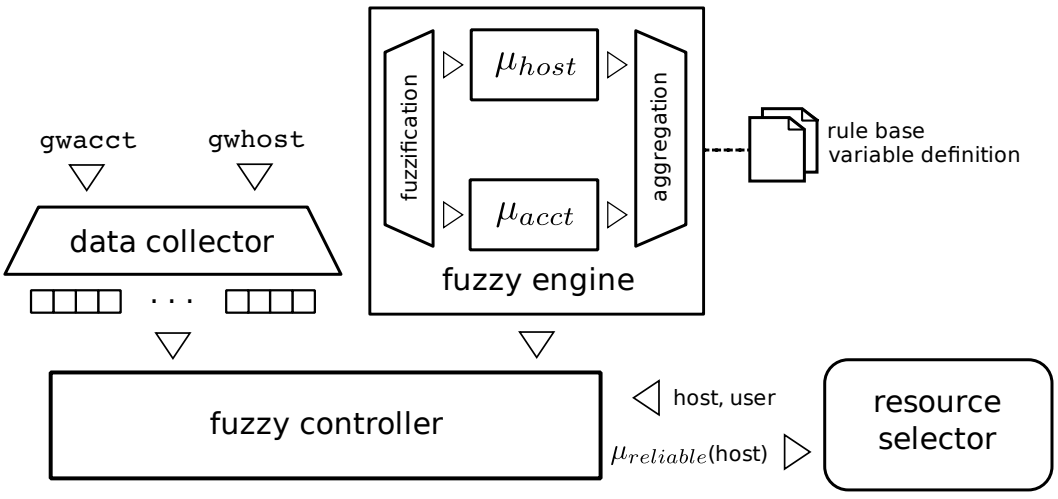


Figura 5.1: Sistema borroso para calcular la fiabilidad

El siguiente algoritmo presenta cómo sería el bucle principal del sistema borroso:

while true
  Esperar orden de GridWay
  $resource \leftarrow$ read()
  $user \leftarrow$ read()
  $template \leftarrow$ read()
  $model \leftarrow$ DataCollector.getResource($resource, user$)
  $reliability \leftarrow$ FuzzySyste.evalueteReliability($template$, $model$)
  write($reliability$)

En la imagen 5.2 se presenta un diagrama de colaboración para obtener una visión más detallada de cómo se relacionan los diferentes módulos durante la ejecución de una petición de GridWay.

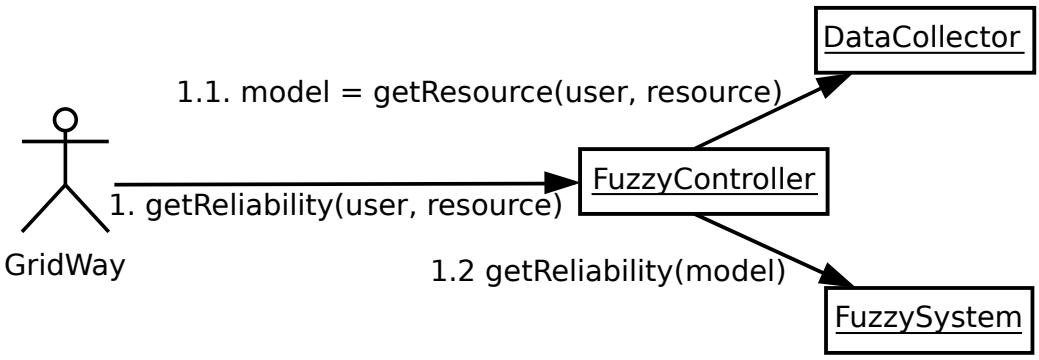


Figura 5.2: Colaboración para atender una petición de GridWay

5.3. Diseño detallado

Una vez descritos el funcionamiento general del sistema y su arquitectura, en esta sección se describen algunos componentes en mayor profundidad, para dar una mejor cobertura a las partes más significativas del sistema borroso.

5.3.1. Comunicación entre el sistema borroso y GridWay

GridWay y el sistema borroso se ejecutan como procesos diferentes en la misma máquina. Es necesario, por tanto, algún método IPC (*Inter-process Communication*) que implemente el siguiente intercambio de información:

1. GridWay envía al sistema borroso los identificadores del recurso y del usuario y la ruta donde está ubicado el *job template* del trabajo.

2. Con estos parámetros, el sistema borroso calcula la fiabilidad y envía el resultado a GridWay.

El flujo de comunicación debe ser bireccional, por lo que, en un primer momento, se optó por utilizar tan un método sencillo como dos tuberías con nombre (*pipes*), una donde GridWay escribiría la información necesaria (`fuzzy_in`) y otra en la que el sistema borroso escribiría el resultado del cálculo de fiabilidad (`fuzzy_out`). Este método, sin embargo, presentó bastante problemas de configuración, tanto a equipos Unix/Linux como Windows, por lo que se descartó.

La opción elegida finalmente fueron los *sockets* de red, por su versatilidad y su portabilidad a cualquier entorno. Un *socket* es cada punto final de un enlace entre dos procesos a través de una red, y que permite comunicación bidireccional. Esta solución necesita de una arquitectura cliente-servidor, de tal forma que el sistema borroso actúa de servidor, minimizando el impacto en el código de GridWay.

5.3.2. Interpretación de reglas borrosas

El `FuzzySystem` es el componente que se encarga de realizar las operaciones lógicas necesarias para obtener el valor de la fiabilidad de un recurso. Es necesario volver a generar tanto las reglas como las funciones de pertenencia de los términos de cada variable lingüística para cada trabajo. Toda esta información se encuentran en el archivo de configuración del trabajo o *job template* y cada trabajo puede definir los parámetros del sistema borroso en función de sus características.

Un *job-template* es un archivo de texto con el formato presentado en el listado 5.1. Este fichero debe incluir la siguiente información necesaria para la configuración de `FuzzySystem`:

1. Definición de las variables lingüísticas y de cada uno de sus términos.
2. Familia de conectores lógicos que pueden ser utilizados.
3. Las reglas que definen los conjuntos μ_{mon} y μ_{acct} mediante proposiciones lógicas borrosas.
4. La fórmula que pondera la influencia de cada uno de los conjuntos en el cálculo de la fiabilidad.

Construcción del analizador

La construcción del sistema lógico debe realizarse examinando el contenido del archivo. Es necesario, por tanto, disponer de un analizador o *parser* que sea capaz de reconocer las secuencias o elementos que definen el lenguaje de los *job templates*.

Existen herramientas, llamadas metacompiladores o generadores de *parsers*, que implementan un analizador tomando como entrada la especificación de un lenguaje. En general, la especificación del lenguaje puede incluir los aspectos léxico, sintáctico y semántico. Uno de los metacompiladores más utilizados que producen código Java es JavaCC[4]. JavaCC genera analizadores descendentes de tipo LL(k) y permite incluir código Java en

el interior de las reglas de expansión de la gramática. De esta forma pueden especificarse las acciones semánticas y mantener en un mismo archivo todos los aspectos del lenguaje.

La gramática para JavaCC del lenguaje de los *job-templates* se muestra en el listado 5.2. Los aspectos más importantes de la gramática son los siguientes:

- En un primer momento se leen las variables lingüísticas, cada una delimitada por las palabras clave `FUZZIFY` y `END_FUZZIFY`. Esa declaración incluye el nombre de la variable y la definición de uno o más términos. Cada término viene especificado por un nombre y tres puntos que determinan su función de pertenencia triangular.

- La última parte viene delimitada por `RULES` y `END_RULES`. Este bloque contiene, por este orden, la definición de la familia de conectivas lógicas, que será un tipo de `MINIMUM`, `PRODUCT` o `LUKASIEWICZ`, la definición de los conjuntos `MON` y `ACCT` y, por último, la fórmula para combinar ambos valores.

```
TOKEN:
{
    < #DIGIT: ["0"-"9"] >
  | < #LETTER: ["A"-"Z"] | ["a"-"z"] >
  | < #SYMBOLS: "_" >
  | < #POINT: "." >
  | < RULES: "RULES" >
  | < END_RULES: "END_RULES" >
  | < AND: "AND" >
  | < OR: "OR" >
  | < NOT: "NOT" >
  | < IS: "IS" >
  | < MON: "MON" >
  | < ACCT: "ACCT" >
  | < REL: "REL" >
  | < FUZZIFY: "FUZZIFY" >
  | < END_FUZZIFY: "END_FUZZIFY" >
  | < TERM: "TERM" >
  | < FAMILY: "FAMILY" >
  | < MINIMUM: "MINIMUM" >
  | < PRODUCT: "PRODUCT" >
  | < LUCASIEWICZ: "LUCASIEWICZ" >
  | < VARIABLE: (<LETTER>)+ (<LETTER> | <DIGIT> | <SYMBOLS
    >)* >
  | < OPENPAR: "(" >
  | < CLOSEPAR: ")" >
  | < COMMA: "," >
  | < SEMICOLON: ";" >
  | < ASSIGN: ":=" >
  | < EQUAL: "=" >
```

```
  | < PLUS: "+" >
  | < TIMES: "*" >
  | < CONSTANT: (<DIGIT>)+ (<POINT> (<DIGIT>)+)? >
}

void fuzzyFile():
{
  LinguisticVariable variable = null;
}
{
  (
    variable = fuzzifyDefinition()
    {this.fuzzySystem.add(variable);}
  )*
  gridwayDefinition()
}

LinguisticVariable fuzzifyDefinition() :
{
  LinguisticVariable variable = null;
  String name = "";
  Point2D.Double min = null;
  Point2D.Double mid = null;
  Point2D.Double max = null;
}
{
  <FUZZIFY><VARIABLE>
  {variable = new LinguisticVariable(token.image);}
  (
    <TERM><VARIABLE>{name = token.image;}
    <ASSIGN>
    min = point()
    mid = point()
    max = point()
    <SEMICOLON>
    {variable.add(new LinguisticTerm(name,new
       TriangularMembershipFunction(min,mid,max)));}
  )+
  <END_FUZZIFY>
  {return variable;}
}

Point2D.Double point() :
{
  double x,y;
}
```

```
{
  <OPENPAR><CONSTANT>{x = Double.parseDouble(token.image)
     ;}<COMMA><CONSTANT>{y = Double.parseDouble(token.image
     );}<CLOSEPAR>
  {return new Point2D.Double(x,y);}
}

void gridwayDefinition() :
{

}
{
  <RULES>
  (logicFamily())?
  monitoringRule()
  accountingRule()
  reliabilityRule()
  <END_RULES>
}

void logicFamily() :
{
  FuzzyFamily family = null;
}
{
  <FAMILY><EQUAL>
  (
    <MINIMUM>
    { family = new MinimumFamily(); }
    |
    <PRODUCT>
    { fuzzyFamily = new ProductFamily(); }
    |
    <LUCASIEWICZ>
    { fuzzyFamily = new LucasiewiczFamily(); }
  )
  <SEMICOLON>
  {fuzzySystem.setFuzzyFamily(family);}
}

void monitoringRule() :
{
  RuleNode node = null;
  FuzzyRule rule = null;
}
{
```

```
  <MON><EQUAL>node = ruleOr()<SEMICOLON>
  {
    rule = new FuzzyRule();
    rule.setRootNode(node);
    this.fuzzySystem.setMonitoringRule(rule);
  }
}

void accountingRule() :
{
  RuleNode node = null;
  FuzzyRule rule = null;
}
{
  <ACCT><EQUAL>node = ruleOr()<SEMICOLON>
  {
    rule = new FuzzyRule();
    rule.setRootNode(node);
    this.fuzzySystem.setAccountingRule(rule);
  }
}

RuleNode ruleOr() :
{
  RuleNode last = null;
  RuleNode right = null;
  RuleNode or = null;
}
{
  last = ruleAnd()
  (
    <OR> right = ruleAnd()
    {
      or = new RuleNodeConnectorOr();
      or.addNode(last);
      or.addNode(right);
      last = or;
    }
  )*
  {return last;}
}

RuleNode ruleAnd() :
{
  RuleNode last = null;
  RuleNode right = null;
```

```
  RuleNode and = null;
}
{
  last = term()
  (
    <AND> right = term()
    {
      and = new RuleNodeConnectorAnd();
      and.addNode(last);
      and.addNode(right);
      last = and;
    }
  )*
  {return last;}
}

RuleNode term() :
{
  LinguisticVariable variable = null;
  LinguisticTerm term = null;
  RuleNode node = null;
  boolean negated = false;
}
{
  <OPENPAR>node = ruleOr()<CLOSEPAR>
  {return node;}
  |
  (
     <NOT>
     {negated = true;}
  )?
  <VARIABLE>
  {
    variable = this.fuzzySystem.get(token.image);
    if (variable == null)
    {
      throw new ParseException("The variable " + token.
         image + " is not defined in the model.");
    }
  }
  <IS><VARIABLE>
  {
    term = variable.get(token.image);
    if (term == null)
    {
      throw new ParseException("The term " + token.image +
```

```
            " is not defined for variable " + variable.
            getName() + ".");
      }

      return new RuleNodeTerminal(new FuzzyTerm(variable.
         getName(),negated,term));
    }
  }

  void reliabilityRule() :
  {

  }
  {
    <REL><EQUAL><CONSTANT>
    {
      this.fuzzySystem.setAccountingWeight(Double.
         parseDouble(token.image));
    }
    <TIMES><ACCT><PLUS><CONSTANT>
    {
      this.fuzzySystem.setMonitoringWeight(Double.
         parseDouble(token.image));
    }
    <TIMES><MON><SEMICOLON>
  }
```

Listado 5.2: Gramática del lenguaje de los *job templates*

A medida que avanza el análisis se va creando el árbol de objetos necesarios que conforman las reglas y que permiten evaluar la fiabilidad de los recursos. El código Java necesario ha sido introducido como la parte semántica de las reglas de expansión. Por tanto, una vez completado el análisis, el objeto *FuzzySystem* es totalmente funcional.

El lenguaje FCL

El lenguaje FCL (*Fuzzy Control Language*), estándar *IEC 61131-7*[3], tiene como objetivo poder habilitar la compartición de manera sencilla de proyectos de Control Borroso entre sistemas con interfaz FCL. Este lenguaje, al ser tan especializado y sencillo de entender, permite que alguien con unas pocas nociones sobre control borroso comprenda el script mostrado en la figura 5.3.

Durante la concepción del proyecto, el lenguaje FCL apareció como un estándar que permitía generar *scripts* en un formato sencillo pero potente. Para la construcción de los primeros prototipos se utilizó, además, una librería como JFuzzyLogic[6] que permitía de manera bastante sencilla leer ficheros FCL y construir un sistema de Razonamiento Aproximado. Al avanzar el proyecto, sin embargo, se descartó que se pudiera utilizar un

motor de Razonamiento Aproximado al uso, por lo que se implementó una librería propia de evaluación de proposiciones. No obstante, sí se han aprovechado el conjunto de buenas características del lenguaje FCL, como la sencillez y la fácil comprensión, basando la mayor parte de la sintaxis de los *job templates* en dicho lenguaje.

```
FUNCTION_BLOCK Fuzzy_FB
VAR_INPUT
  temp : REAL;
  pressure : REAL;
END_VAR

VAR_OUTPUT
  valve : REAL;
END_VAR

FUZZIFY temp
  TERM cold := (3, 1)  (27, 0);
  TERM hot := (3, 0  (27, 1);
END_FUZZIFY

FUZZIFY pressure
  TERM low := (55, 1) (95, 0);
  TERM high:= (55, 0) (95, 1);
END_FUZZIFY

DEFUZZIFY valve
  TERM drainage := -100;
  TERM closed := 0;
  TERM inlet := 100;
  ACCU : MAX;
  METHOD : COGS;
  DEFAULT := 0;
END_DEFUZZIFY

RULEBLOCK No1
AND : MIN;
  RULE 1 : IF temp IS cold AND pressure IS low THEN valve
     IS inlet
  RULE 2 : IF temp IS cold AND pressure IS high THEN valve
      IS closed WITH 0.8;
  RULE 3 : IF temp IS hot AND pressure IS low THEN valve
     IS closed;
  RULE 4 : IF temp IS hot AND pressure IS high THEN valve
     IS drainage;
END_RULEBLOCK
```

```
END_FUNCTION_BLOCK
```

Listado 5.3: Ejemplo de sistema borroso en FCL

5.3.3. Recolección de información

El componente `DataCollector` es una parte esencial del sistema. Es el encargado de obtener del entorno la información necesaria para rellenar el modelo de recurso, ya sea directamente del usuario, de GridWay o un banco de pruebas.

La clase `DataCollector` es una clase abstracta de la que deben heredar las clases que realmente implementan la funcionalidad. Durante el desarrollo de este proyecto se han implementado tres `DataCollector` diferentes, dependiendo del entorno y los requisitos de cada caso. En un primer momento se utilizó el `CommandLineDataCollector` que requería los datos al usuario durante la ejecución. Este entorno interactivo se utilizó para las primeras pruebas del sistema. En un segundo paso se implementó el `GridWayDataCollector` que es capaz de obtener de GridWay toda la información necesaria. En la siguiente sección se detalla el comportamiento de esta clase. La tercera implementación, `GridSimDataCollector`, fue necesaria al preparar un banco de pruebas virtual donde poder someter al planificador borroso a una carga de trabajo real. Este recolector es presentado con detalle en el capítulo 6 donde se presenta la arquitectura del banco de pruebas.

La clase `GridWayDataCollector`

En su entorno definitivo, el sistema difuso debe ser capaz de comunicarse con GridWay para tener acceso a toda la información de un recurso, ya sea información de monitorización o datos de contabilidad. GridWay ofrece dos comandos con los que permite obtener toda esta información: `gwhost` para conocer las características y el estado del recurso; `gwacct` para toda la información de contabilidad sobre ejecuciones pasadas. La clase `GridWayDataCollector` es capaz de interpretar esta salida y, combinando todos esos datos, ser capaz de rellenar el modelo de recurso. Las operaciones necesarias para generar los diferentes campos del modelo de recurso vienen especificadas en un fichero de configuración externo llamado `resource.properties` que tiene la siguiente forma:

```
# HOST
cpu_speed = CPU_MHZ/CPU_MHZ_MAX
nodes_total = NODECOUNT/NODECOUNT_MAX
nodes_available = FREENODECOUNT/NODECOUNT

# TIMES
transfer_time = XFR/(EXE + XFR + SUSP)
execution_time = EXE/(EXE + XFR + SUSP)
suspension_time = SUSP/(EXE + XFR + SUSP)

# JOBS
```

```
success_rate = SUCC/TOTS
error_rate = ERR/TOTS
suspension_rate = SUSPJ/TOTS

# HOST
cpu_speed = CPU_MHZ/CPU_MHZ_MAX
nodes_total = NODECOUNT/NODECOUNT_MAX
nodes_available = FREENODECOUNT/NODECOUNT

# TIMES
transfer_time = XFR/(EXE + XFR + SUSP)
execution_time = EXE/(EXE + XFR + SUSP)
suspension_time = SUSP/(EXE + XFR + SUSP)

# JOBS
success_rate = SUCC/TOTS
error_rate = ERR/TOTS
suspension_rate = SUSPJ/TOTS
```

Listado 5.4: El archivo de configuración `resource.properties`

En él se asigna una expresión a cada una de las variables del modelo de recurso, indicando cómo será calculado su valor a partir de la información proporcionada por GridWay. Para analizar las expresiones algebraicas que aparecen en la parte derecha se contruyó un *parser* siguiendo el mismo procedimiento descrito en la sección 5.3.2. La especificación de la gramática para el metacompilador JavaCC se muestra en el listado 5.5:

```
TOKEN:
{
  < CONSTANT: (<DIGIT>)+ (<POINT> (<DIGIT>)+)? >
  | < VARIABLE: (<LETTER>)+ (<LETTER> | <DIGIT> | <SYMBOLS
     >)* >
  | < HOUR: <DIGIT><DIGIT><COLON><DIGIT><DIGIT><COLON><
     DIGIT><DIGIT> >
  | < PLUS: "+" >
  | < MINUS: "-" >
  | < MULTIPLY: "*" >
  | < DIVIDE: "/" >
  | < MODULE: "%" >
  | < OPENPAR: "(" >
  | < CLOSEPAR: ")" >
  | < #DIGIT: ["0"-"9"] >
  | < #LETTER: ["A"-"Z"] | ["a"-"z"] >
  | < #SYMBOLS: "_" >
  | < #POINT: "." >
  | < #COLON: ":" >
```

```
}

// expr --> sum
double expr() throws java.text.ParseException:
{
  double result;
}
{
  result = sum()
  {return result;}
}

// sum --> product (<PLUS> product | <MINUS> product)*
double sum() throws java.text.ParseException:
{
  double result;
  double temp;
}
{
  result = product()
  (
    <PLUS> temp = product()
    {result += temp;}
    |
    <MINUS> temp = product()
    {result -= temp;}
  )*
  {return result;}
}

// product --> term (<MULTIPLY> term | <MODULE> term | <
   DIVIDE> term)*
double product() throws java.text.ParseException:
{
  double result;
  double temp;
}
{
  result = term()
  (
    <MULTIPLY> temp = term()
    {result *= temp;}
    |
    <MODULE> temp = term()
    {result %= temp;}
    |
```

```
    <DIVIDE> temp = term()
    {result /= temp;}
  )*
  {return result;}
}

// term --> <OPENPAR> sum <CLOSEPAR> | <CONSTANT> | <
   VARIABLE>
double term() throws java.text.ParseException:
{
  Token token;
  double result;
}
{
  <OPENPAR> result = sum() <CLOSEPAR>
  {return result;}
  |
  token = <CONSTANT>
  {return Integer.parseInt(token.image);}
  |
  token = <HOUR>
  {return this.date_parser.parse(token.image).getTime()
     /1000;}
  |
  token = <VARIABLE>
  {
    String variable_value = this.symbols_table.getProperty
       (token.image);
    if (variable_value.indexOf(':') != -1)
    {
      result = this.date_parser.parse(variable_value).
         getTime()/1000;
    }
    else
    {
      result = Integer.parseInt(variable_value);
    }
    return result;
  }
}
```

Listado 5.5: Gramática del lenguaje de `resource.properties`

Capítulo 6

Resultados

En este capítulo se presentan y analizan los resultados obtenidos con el planificador borroso, viendo cómo se comporta en relación al planificador implementado actualmente por GridWay. También se analiza el comportamiento de las diferentes familias lógicas implementadas por el sistema borroso.

Al comienzo de este capítulo se describe el banco de pruebas desarrollado con el fin de analizar el rendimiento del nuevo algoritmo de selección de recursos. En primer lugar se introducen los simuladores Grid y por qué son una alternativa aceptable a probar directamente sobre un entorno Grid real. Seguidamente se presenta brevemente la librería GridSim, viendo sus principales características. Por último, se expone la arquitectura del banco de pruebas detallando sus partes más importantes.

La segunda parte de este capítulo describe los experimentos realizados y los resultados obtenidos. Posteriormente se analizan y se presentan las conclusiones que se derivan de los resultados obtenidos.

6.1. Implementación del banco de pruebas

La investigación de nuevas técnicas y algoritmos de planificación en entornos Grid conlleva una dificultad añadida respecto a otros entornos de ejecución. Dicha dificultad radica en el tiempo físico necesario para ejecutar cada uno de los experimentos. Esto se debe a que los sistemas Grid están diseñados para ejecutar aplicaciones con una gran carga computacional, y por lo tanto todas las investigaciones sobre nuevas técnicas de planificación en Grid deben llevar asociadas ejecuciones de aplicaciones con cargas computacionales elevadas. De esta manera, el tiempo necesario para obtener los resultados de la investigación se incrementa considerablemente llegando a durar días, e incluso semanas. Además, disponer de un entorno Grid real para hacer pruebas de algoritmos de planificación no es algo sencillo dada la infraestructura y el número de organizaciones involucradas.

Por otra parte, el dinamismo inherente al Grid dificulta el análisis de las pruebas debido a la continua evolución del entorno de ejecución. Esto hace que no pueda asegurarse que dos ejecuciones diferentes hayan contado con un entorno de ejecución idéntico.

Para resolver estos problemas, en los últimos años han aparecido herramientas que permiten al desarrollador diseñar infraestructuras Grid virtuales adaptadas a sus necesidades, eliminando así la necesidad de disponer de un entorno Grid real. Estas herramientas que permiten la creación de entornos Grid muy diversos en los cuales simular la ejecución de aplicaciones con una gran carga computacional en un tiempo mucho menor. Estos simuladores de entornos Grid permiten al usuario diseñar una gran cantidad de infraestructuras Grid virtuales y por lo tanto son muy útiles como banco de pruebas para la ejecución de experimentos. Además, permiten que dos experimentos diferentes sean ejecutados sobre el mismo entorno, simplificando el análisis de los resultados.

En el resto de la sección se divide en dos partes. Primero se introduce brevemente la librería GridSim que ha sido elegida para implementar el banco de pruebas de este proyecto. La segunda parte describe la arquitectura y la implementación del banco de pruebas utilizado para este proyecto.

6.1.1. GridSim

El simulador GridSim, creado por Rajkumar Buyya y M. Manzur Murshed [28], es una librería escrita en el lenguaje de programación Java que permite diseñar y simular recursos Grid heterogéneos. GridSim permite modelar todo el entorno del Grid y probar algoritmos de planificación. El entorno incluye tanto los recursos, como a los usuarios con diferentes requisitos del Grid (en términos de aplicación y QoS).

6.1.2. Arquitectura del banco de pruebas

El objetivo principal del simulador consiste en servir como herramienta para analizar el comportamiento del planificador de tareas, antes de su ejecución en un entorno real Grid heterogéneo y distribuido. Para ello, el banco de pruebas debe ser capaz de simular un entorno Grid completo.

La figura 6.1 presenta la arquitectura del banco de pruebas, mostrando cuáles son los componentes más relevantes. Estos componentes son clases o jerarquías de clases Java que están diseñadas para que el comportamiento del entorno simulado se asemeje lo más posible a un Grid.

Algunas de las ideas de la arquitectura están cogidas del proyecto Alea[23]. Alea es un simulador basado en GridSim que extiende la funcionalidad proporcionada por esta librería. Particularmente, tanto la concepción como el código de las clases `UserWorkload`, para la lectura de archivos de cargas de trabajo, y `ResourceFailureLoader`, para cargar archivos de fallos de recursos, están bastante inspiradas en esta librería.

Los siguiente apartados de esta sección describen en detalle cada uno de los elementos que forman el entorno simulado por el banco de pruebas.

Trabajos y usuarios

La carga de trabajo que es ejecutada en el banco de pruebas es leída de un fichero externo en formato `.swf`, o Standard Workload Format. Los archivos en este formato

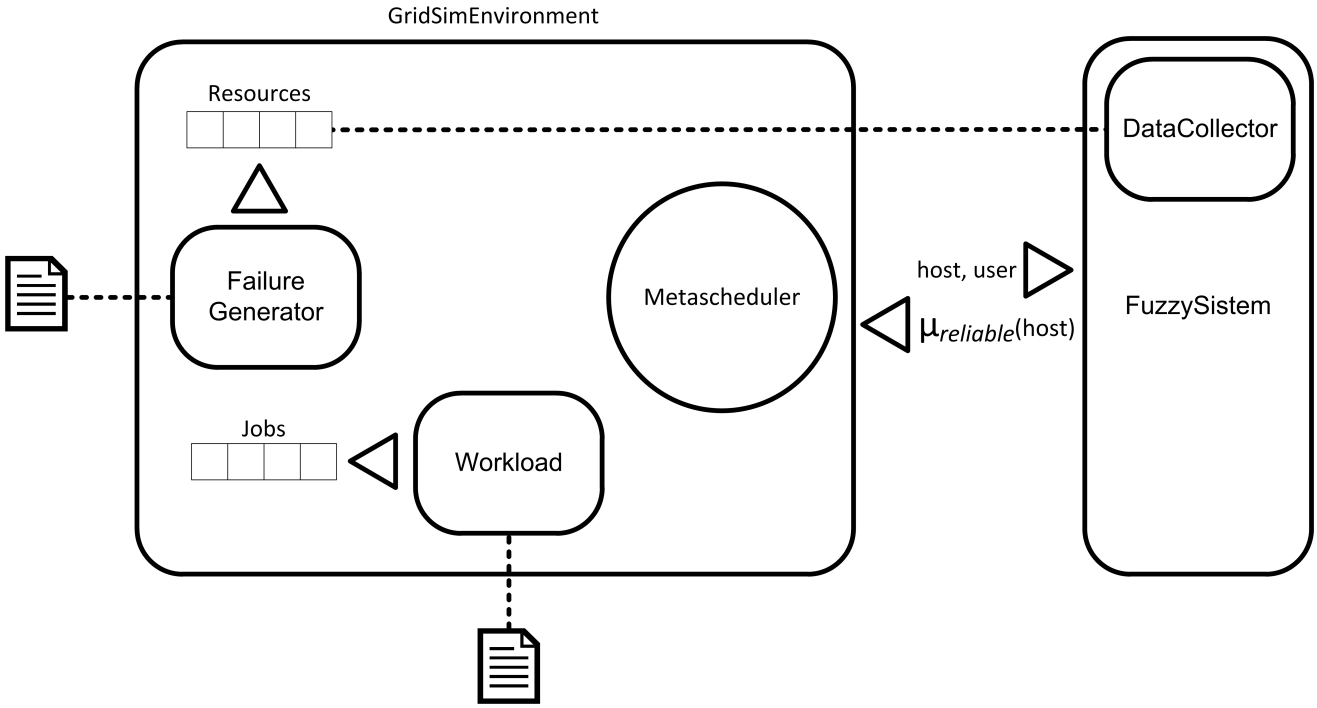


Figura 6.1: Arquitectura del banco de pruebas

son de texto ASCII y definen todos los parámetros de la ejecución de un trabajo en una única línea. En este proyecto actualmente son utilizados los campos de *tiempo de emisión*, *tiempo de ejecución* y *número de procesadores*.

El primer campo es utilizado, *tiempo de emisión*, es necesario para saber en qué momento de la simulación ese trabajo debe ser lanzado a ejecución. En un sistema Grid real un usuario puede lanzar un trabajo en cualquier momento, es decir, el planificador no sabe en qué momento va a tener que planificar un trabajo ni la carga de trabajo futura. Este comportamiento es reproducido lanzando un evento de nuevo trabajo justo en el momento de la simulación indicado en el archivo.

Los otros dos campos, *tiempo de ejecución* y *número de procesadores*, son combinados para estimar el tamaño del trabajo y la carga que supondrá al recurso.

Recursos

Los recursos son un elemento esencial de cualquier entorno de Computación Distribuída. En este banco de pruebas, los recursos son creados por GridSim como entidades de simulación (pueden enviar y recibir eventos) que tienen asociados una política interna, a modo de planificador local, y unas características. Tanto la política como las características han sido extendidas para añadir funcionalidad adicional necesaria dando lugar a las clases `GridwayPolicy` y `GridwayResourceCharacteristics`. Estas nuevas funcionalidades son: (i) el recurso debe ser capaz de almacenar toda la información de contabilidad y de monitarización y (ii) que el recurso pueda almacenar los momentos programados en

los que estará caído para simular fallos. La figura 6.2 muestra la jerarquía de clases y sus relaciones.

La clase `GridwayResourceCharacteristics` extiende a la clase `ResourceCharacteristics` para ser capaz de almacenar toda la información de monitorización y contabilidad de un recurso que es necesaria como entrada al Sistema Borroso. Esto supone, por tanto, todos los campos del modelo abstracto de recurso.

Por su parte, la clase `GridwayPolicy` hereda de la clase abstracta `AllocPolicy` y es, principalmente, la encargada de manejar la ejecución los trabajos que llegan a un recurso. Los trabajos pasarán a ejecución si el recurso está libre o añadidos a la cola de suspendidos en caso contrario. Esta cola se maneja como con un algoritmo FIFO. Otra de las tareas asignadas a esta clase es la de gestionar los fallos y caídas del recurso. En caso de llegar un fallo, tanto los trabajos suspendidos como en ejecución serán marcados como erróneos y el planificador deberá migrar cada trabajo hacia otro recurso. Esta clase utiliza toda la información sobre el estado del recurso y de los trabajos asignados para mantener actualizados los valores del objeto `GridwayResourceCharacteristics`.

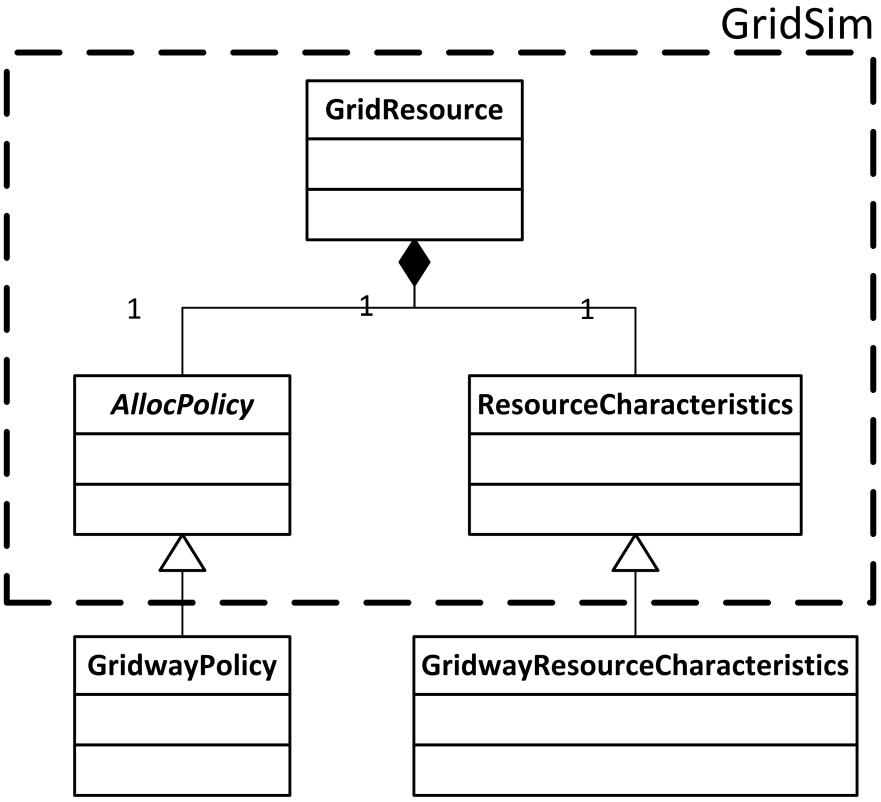


Figura 6.2: Componentes de `GridResource` personalizados para el banco de pruebas

Generación de fallos en los recursos

Para simular la componente dinámica y aleatoria del Grid tienen que existir los fallos y cortes de servicio en los recursos. Estos fallos son leídos de un fichero de texto externo, `errors.properties`, que presenta líneas de la forma:

nombre = inicio, fin

donde *nombre* es el nombre del recurso, *inicio* el momento de la simulación donde se produce el corte y *fin* el instante donde el recurso vuelve a funcionar normalmente. Un ejemplo de este fichero se muestra en el listado 6.1.

```
Resource_0 = 150000, 180000
Resource_1 = 7500, 100000
Resource_2 = 200000, 750000
Resource_3 = 50, 250000
```

Listado 6.1: El archivo de definición de errores `errors.properties`

Una vez leídos estos fallos, son cargados en los recursos y es la política `Gridway-Policy` la que comprueba repetitivamente si el tiempo de simulaicón se encuentra dentro del intervalo de corte de servicio.

Recolector de información

La complejidad del recolector de información en este sistema, implementado en la clase `GridSimDataCollector`, es bastante menor que el descrito en la sección 5.3.3 para el entorno real. Como ha sido mencionado anteriormente, toda la información necesaria ha sido compilada y recogida previamente por `GridwayPolicy` en un objeto del tipo `GridwayResourceCharacteristics`. Por lo tanto, `GridSimDataCollector` únicamente debe rellenar los campos del modelo de recurso con los datos proporcionados en ese objeto de características.

Metaplanificador

El metaplanificador del banco de pruebas es el encargado de enviar cada trabajo a un recurso concreto basándose en los resultados de la medida de fiabilidad proporcionada por el sistema borroso. Este planificador está implementado por la clase `GridSimScheduler` que hereda de `GridSim` toda la funcionalidad necesaria para ser una entidad de simulación y poder gestionar eventos.

Básicamente, esta clase está a la espera de recibir eventos desde `UserWorkload` indicando que un nuevo trabajo ha sido lanzado a ejecución. Una vez recibido el evento, el metaplanificador comienza a analizar la fiabilidad de todos los recursos disponibles para poder planificar el trabajo. Para cada recurso se envían, a través del *socket*, la identidad del recurso y del usuario que ha lanzado el trabajo. Una vez terminado el cálculo, el sistema borroso devolverá el valor por el mismo canal de comunicación. Si este resultado es mejor que el mejor resultado obtenido hasta el momento, este recurso pasa a ser el seleccionado. El trabajo será enviado al recurso que quede selecionado al final.

6.2. Resultados

Esta sección define en primer logar el procedimiento de pruebas que se ha seguido a la hora de comprobar el funcionamiento del sistema borroso. La segunda parte presenta y analiza los resultados obtendos durante la batería de tests.

6.2.1. Procedimiento de pruebas

El procedimiento utilizado para obtener los resultados presentados en este capítulo es lanzar cargas de 10, 50 100, 250, 500, 1000 y 2000 trabajos al banco de pruebas y comprobar el número de errores y suspensiones se producen durante la ejecución. Esta prueba es repetida para las tres familias lógicas implementadas en el sistema borroso: `MINIMUM`, `PRODUCT` y `LUKASIEWICZ`. Hay que destacar que todos los trabajos pertenecientes a la misma batería de tests utilizarán las mismas reglas de definición de los conjuntos *mon*, *acct* y *reliability* para poner a prueba el comportamiento general de la medida borrosa de fiabilidad.

Además, estos resultados se comparan con el que se obtendría utilizando una heurística *crisp*. La regla implementada para calcular la fiabilidad de esta manera también hace uso de los datos contenidos en el modelo de recurso, pero realizando únicamente operaciones aritméticas para expresar los requisitos del trabajo:

$$reliability = \frac{1}{2}(nodes\_available + cpu\_speed) + \frac{1}{2}(success\_rate - error\_rate)$$

Resultados de la primera prueba

La primera prueba se realiza utilizando la lógica descrita en 6.2. La carga de trabajo lanzada al banco de pruebas está tomada del archivo de datos de entrada `NASAiPSC-19932.1cln.swf`. Los resultados generados están presentados en las gráficas de la figura 6.3.

```
MON = nodes_available IS high;
ACCT = suspension_rate IS low AND success_rate IS high
   AND error_rate IS low;
REL = 0.67 * ACCT + 0.33 * MON;
```

Listado 6.2: Reglas utilizadas para la pruebas 1 y 2

Resultados de la segunda prueba

La segunda prueba se realiza utilizando la misma lógica usada para la prueba anterior. Sin embargo, se modifica el fichero de carga de trabajo, pasando a utilizar `DAS2fs02003-1.swf.gz` como fichero de entrada. Las gráficas en 6.4 reflejan los resultados obtenidos tras la ejecución de esta batería de tests.

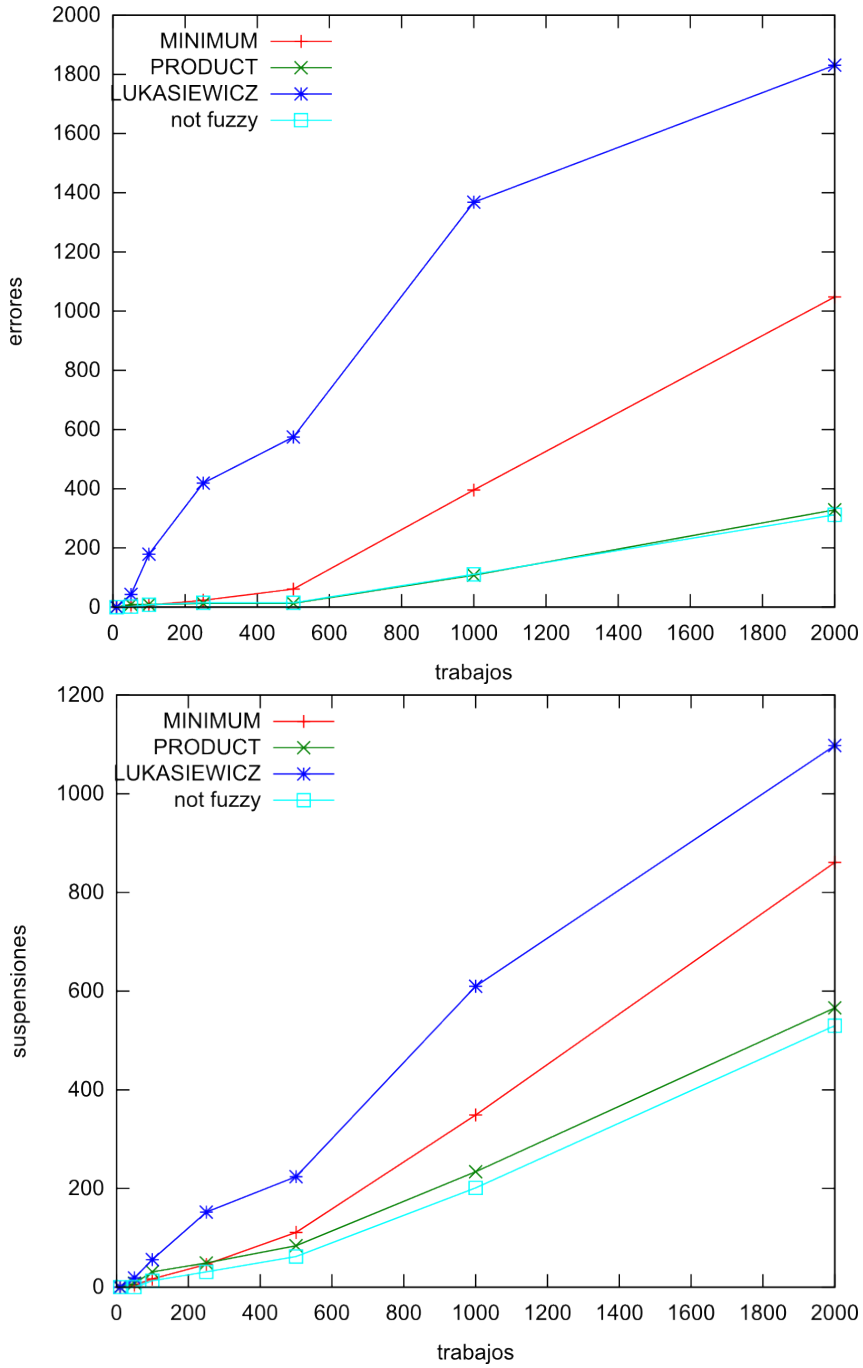


Figura 6.3: Resultados obtenidos durante la primera prueba

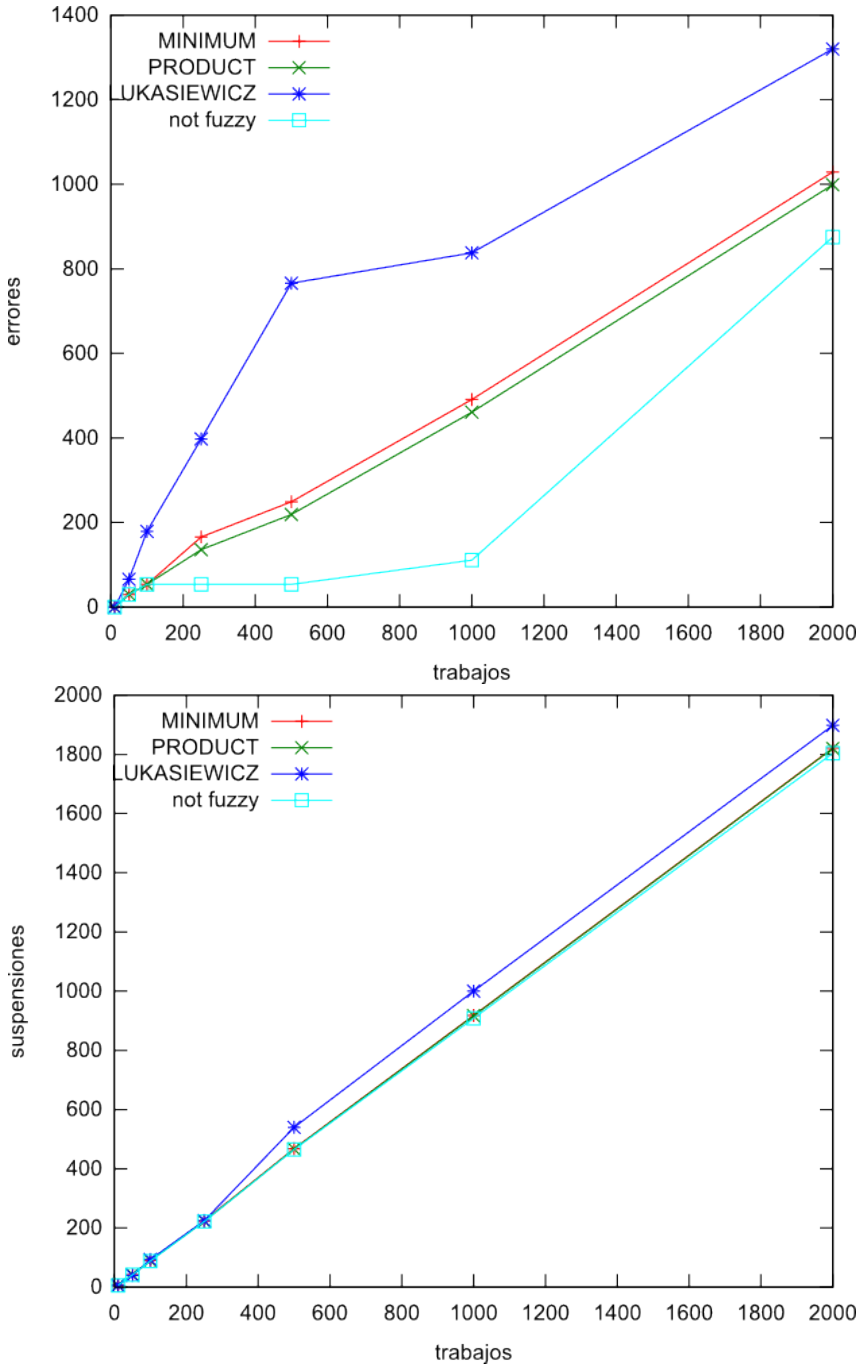


Figura 6.4: Resultados obtenidos durante la segunda prueba

Resultados de la tercera prueba

Esta prueba ejecuta introduce un cambio en la lógica, complicando la regla de `ACCT` para intentar sacar más partido del motor de lógica borrosa. Las nuevas reglas utilizadas se exponen en el listado 6.3. Para esta ejecución se hace uso del archivo de carga de trabajo `NASAiPSC19932.1cln.swf`. Los resultados generados por el banco para esta prueba están presentados la figura 6.5.

```
MON = nodes_available IS high;
ACCT = (suspension_rate IS low OR suspension_time IS low)
   AND (success_rate IS high OR error_rate IS low);
REL = 0.5 * ACCT + 0.5 * MON;
```

Listado 6.3: Reglas utilizadas para la pruebas 3 y 4

Resultados de la cuarta prueba

En último lugar se ejecuta otra batería de tests utilizando la lógica descrita para la anterior prueba. La carga de trabajo utilizada viene definida en el archivo `DAS2fs02003-1.swf.gz`. Los resultados obtenidos pueden verse en la figura 6.6.

6.3. Conclusiones

Estos resultados ponen de manifiesto la importancia de seleccionar adecuadamente la familia lógica que se quiere utilizar, ya que el rendimiento obtenido varía muy significativamente. En este sentido, hay que destacar a la familia lógica de `LUKASIEWICZ` como la eficiencias más bajas ha registrado tras la ejecución de estas pruebas.

Otra de las conclusiones que se extraen viendo la distribució de los trabajos observada en la tabla 6.1. Esta tabla muestra un detalle de la ejecución de 1000 trabajos en el simulador utilizando la lógica anterior. Se ve claramente que una medidas acumulativas del tipo *error_rate*, *suspension_rate* y *success_rate* conducen a un uso prácticamente nulo de los recursos que fallan en las primeras ejecuciones. El sistema debe añadir campos en el modelo de recurso que representen las últimas n y que permitan volver a intentar una ejecución en un nodo fallido transcurrido un cierto periodo de tiempo.

Observando las gráficas parece evidente que la medida nítida casi siempre se encuentra muy próxima, o incluso supera, al mejor resultado obtenido por el sistema borroso. Esto puede deberse a varias razonas. Una de las razones más factibles que pueden esgrimirse es que el entorno generado no tiene el grado de dinamismo de un Grid real, ya que los recursos tienen únicamente un intervalo en el que están caídos. Dotar al banco de pruebas de mayor dinamismo sería uno de las mejoras a abordar en un futuro próximo.

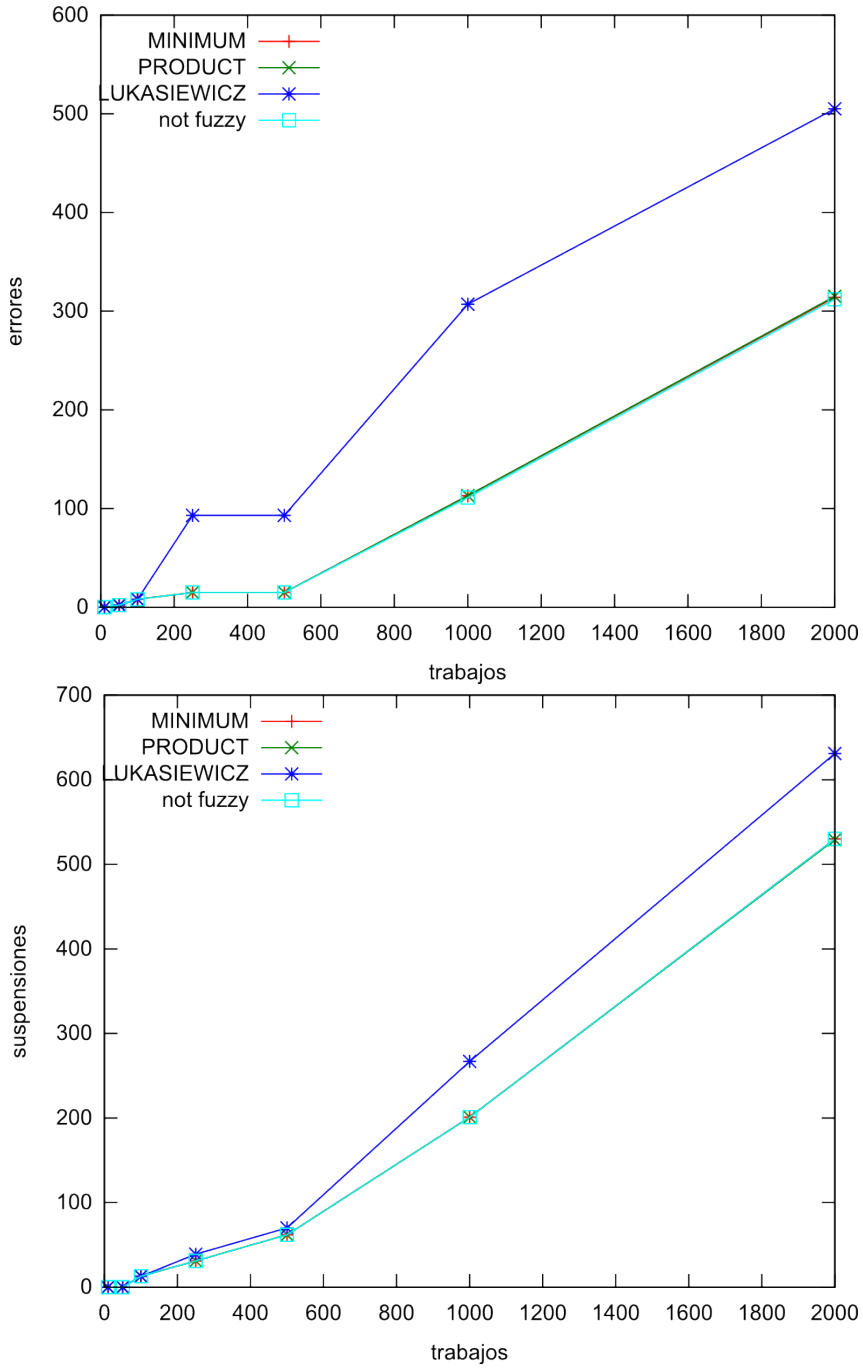


Figura 6.5: Resultados obtenidos durante la cuarta prueba

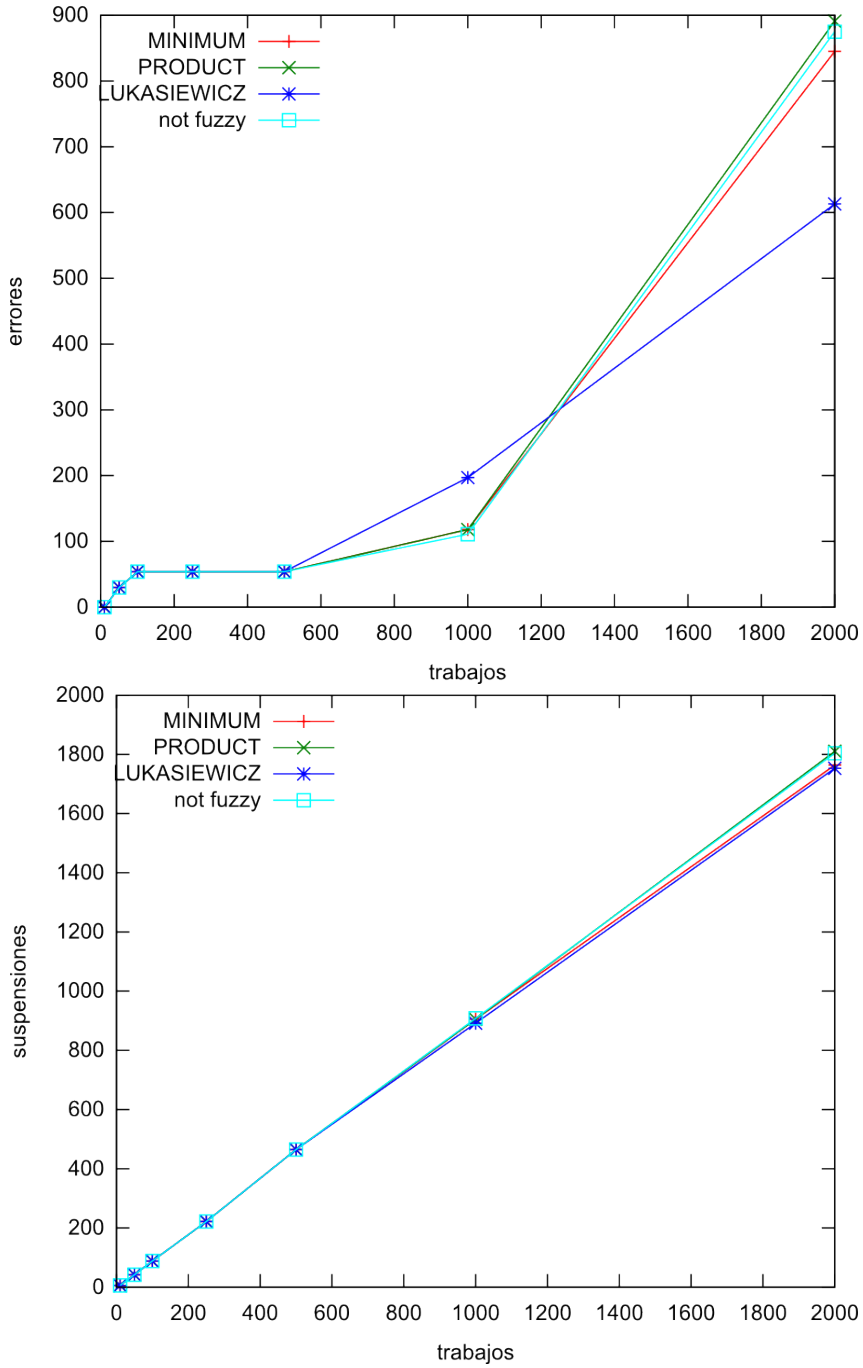


Figura 6.6: Resultados obtenidos durante la cuarta prueba

recurso	ARR	SUC	SUS	ERR
Resource_0	436	425	177	11
Resource_1	161	158	47	3
Resource_2	198	43	10	155
Resource_3	1	0	0	1
Resource_4	2	0	0	2
Resource_5	1	0	0	1
Resource_6	582	364	109	218
Resource_7	8	8	4	0
Resource_8	3	0	0	3
Resource_9	4	2	2	2
total	1396	1000	349	396

Cuadro 6.1: Detalle de la distribución de trabajos

6.3.1. Limitaciones conocidas

El banco de pruebas construído cuenta con una serie de limitaciones que han simplificado el entorno de ejecución. Estas limitaciones ha sido necesario incluirlas para ir aumentando la complejidad del problema de forma paulatina a medida que todo el sistema borroso avance en madurez.

Las limitaciones más importantes son:

- La capacidad de procesamiento de todos los nodos es la misma
- Los trabajos operan con un *job template* común
- La transmisión de los trabajos por la red no está simulada, así como las distintas topologías de red
- El intervalo de fallo de un recurso es único
- En la implementación actual, todos los trabajos son generados por el mismo usuario y no hay diferencias de políticas según el usuario en los recursos

Al no existir diferencias en los tiempos de transmisión y no haber diferencias en la potencia de procesamiento de los recursos, la información de monitorización pasa en este primer momento a un segundo plano. El problema que se está simulando y evaluando en un primer momento es la capacidad del sistema borroso para distribuir la carga convenientemente en función de las suspensiones y los errores que se producen en el entorno virtual del banco de pruebas.

Capítulo 7

Principales aportaciones y trabajo futuro

Este capítulo resume las principales aportaciones presentadas en esta memoria, así como el trabajo futuro que se origina desde este punto. El capítulo se divide en dos secciones, la sección 7.1 realiza un resumen de las ideas aportadas y presenta las publicaciones relacionadas con esta memoria. Y la sección 7.2, describe el trabajo futuro que se origina a partir de las investigaciones llevadas a cabo durante el desarrollo de este proyecto.

7.1. Aportaciones

En esta memoria se presenta una forma novedosa para seleccionar recursos en el metaplanificador GridWay, basada en Lógica Borrosa. Esta nueva heurística se basa en equiparar la fiabilidad de un recurso como su grado de pertenecia al conjunto borroso *reliable*. Este conjunto es construído en base a unas reglas definidas por el usuario utilizando las variables presentes en el modelo de recurso. Los datos para obtener el valor final de fiabilidad de un recurso son proporcionados por los servicios de información e incluyen tanto datos de monitorización y de contabilidad.

Utilizar un sistema borroso para hacer frente al dinamismo y la incertidumbre en un entorno Grid permite hacer frente a la complejidad de manerar más cualitativa que cuantitativa. En particular, el sistema para la selección de recursos en base a la medida borrosa de fiabilidad mejora el sistema actual de dos formas: (i) permite al sistema manejar los diferentes grados de certidumbre de la información, (ii) utiliza un lenguaje para las reglas de selección de recursos lo suficientemente cercano al lenguaje natural para que el operador pueda razonar de la misma manera que lo hace normalmente, permitiendo así obtener un máximo rendimiento de su conocimiento.

Algunas de estas aportaciones ya fueron presentadas en el artículo:

G.P. Santos, L. Garmendia, R.S. Montero, and V. Lopez. Fuzzy system to evaluate resources reliability in a grid environment. In *Proceedings of the 4th International ISKE Conference on Intelligent Systems and Knowledge Engineering*, pages 357–362. World Scientific, November 2009.

El material de presentación relativo a este artículo se incluye en el Apéndice B.

7.2. Trabajo futuro

Los tests realizados al planificador han dado resultados prometedores, aunque no han sido extensivos. Como línea investigadora más clara que se presenta en el futuro, ir paliando alguna de las limitaciones de los tests enumeradas en la sección 6.3.1. Otro paso más allá, una vez completado el anterior sería continuar con las pruebas en entornos reales, intentando verificar si realmente si el sistemas recorta realmente el número de saltos.

Existen todavía algunos puntos del sistema que pueden ser desarrollados. Como se ha visto, el problema del dinamismo del Grid hace que el conjunto de recursos vaya variando. Para que la definición de los términos de las variables lingüísticas se mantenga significativa debe adaptarse a las características de los recursos disponibles en cada momento en el Grid. La idea es poder discriminar los recurso según su grado de pertenencia a alguno de los términos de la variable lingüística. No interesa que se encuentren todos los valores concentrados dentro del mismo término, o agrupados únicamente en un par de términos. Por lo tanto, en el futuro, técnicas de adaptación de las funciones de pertenencia deben ser consideradas.

Otro de los puntos de mejora, como se ha visto al analizar los resultados, sería añadir en el modelo de recurso información sobre las últimas ejecuciones. Esta información puede ser más interesante en un entorno tan cambiante como el Grid que la que proporciona el histórico total de ejecuciones llevadas a cabo por el recurso.

Apéndice A

Modificaciones a GridWay

A.1. Introducción

Los cambios necesarios en el código de GridWay son mínimos. Se necesita añadir la funcionalidad para gestionar la comunicación con el sistema borroso y modificar la evaluación de los recursos para que se haga utilizando la medida de fiabilidad. Los cambios están muy localizados y se centran únicamente en tres archivos:

- `gw_scheduler.h`
- `gw_scheduler_common.c`
- `gw_scheduler_hosts.c`

Las siguientes secciones describen en profundidad los cambios introducidos en cada uno de los ficheros. La versión de GridWay que ha sido utilizada durante el desarrollo de este proyecto es GridWay 5.4.0.

A.2. Cambios en `gw_scheduler.h`

En el archivo de cabecera `include/gw_scheduler.h` se declaran las funciones de apertura y cierre de la comunicación con el sistema borroso, que se llamarán durante la inicialización y la finalización del bucle del planificador.

```
--- gw-5.4.0/include/gw_scheduler.h
+++ gw-5.4.0-modified/include/gw_scheduler.h
@@ -234,5 +234,9 @@

 void gw_scheduler_job_policies(gw_scheduler_t * sched);

+// Communication with fuzzy scheduler
+int open_communication_with_fuzzy_scheduler();
```

```
+int close_communication_with_fuzzy_scheduler();
+
 #endif /*GW_DM_SCHEDULER_H_*/
```

Listado A.1: Parche para `gw_scheduler.h`

A.3. Cambios en `gw_scheduler_common.c`

El hilo principal del planificador de GridWay debe ser modificado para abrir y cerrar las comunicaciones con el sistema borroso. Para ello `src/scheduler/gw_scheduler_common.c`

```
--- gw-5.4.0/src/scheduler/gw_scheduler_common.c
+++ gw-5.4.0-modified/src/scheduler/gw_scheduler_common.c
@@ -142,6 +142,8 @@
                                (time_t) (sched.sch_conf.
                                   window_size * 86400);

  sched.next_host_window = the_time + (time_t) 86400;
+
+ open_communication_with_fuzzy_scheduler();

      gw_scheduler_print('I',"Scheduler successfully
         started.\n");

@@ -362,6 +364,8 @@
              gw_scheduler_print('E',"Unknown action from
                 core %s\n.",act);
          }
      }
+
+ close_communication_with_fuzzy_scheduler();

      if (error == 0)
          fclose(fd_log);
```

Listado A.2: Parche para `gw_scheduler_common.c`

A.4. Cambios en `gw_scheduler_hosts.c`

```
--- gw-5.4.0/src/scheduler/gw_scheduler_hosts.c 2007-06-13
    15:08:30.000000000 +0200
+++ gw-5.4.0-modified/src/scheduler/gw_scheduler_hosts.c
   2011-09-01 13:14:44.529465200 +0200
```

```
@@ -32,6 +32,75 @@
 /* ------------------------------------------------ */
 /* ------------------------------------------------ */

+#include <errno.h>
+static FILE * inpipe;
+static FILE * outpipe;
+
+#define IN_PIPE_PATH "/home/german/gridway/var/fuzzy_out"
+#define OUT_PIPE_PATH "/home/german/gridway/var/fuzzy_in"
+
+int open_communication_with_fuzzy_scheduler()
+{
+ inpipe = fopen(IN_PIPE_PATH,"r");
+ if (inpipe)
+ {
+   gw_scheduler_print('I',"Open %s for reading... OK\n",
   IN_PIPE_PATH);
+ }
+ else
+ {
+   gw_scheduler_print('I',"Open %s for reading... FAILED\
   n",IN_PIPE_PATH);
+   perror(NULL);
+   return -1;
+ }
+
+ outpipe = fopen(OUT_PIPE_PATH,"w");
+ if (outpipe)
+ {
+   gw_scheduler_print('I',"Open %s for writing... OK\n",
   OUT_PIPE_PATH);
+ }
+ else
+ {
+   gw_scheduler_print('I',"Open %s for writing... FAILED\
   n",OUT_PIPE_PATH);
+   perror(NULL);
+   return -1;
+ }
+
+ return 0;
+}
+
+
```

```
+static float get_host_reliability(const char * resource,
   const char * user)
+{
+ float reliability = 0.0f;
+
+ //fscanf(inpipe,"%f",&reliability);
+ // 1. Send information to the fuzzy controller
+ fprintf(outpipe,"%s\n",resource);
+ fprintf(outpipe,"%s\n",user);
+ fflush(outpipe);
+
+ // 2. Get resource's reliability value
+ fscanf(inpipe,"%f",&reliability);
+ gw_scheduler_print('I',"Reliability of %s for %s = %f\n"
   ,resource,user,reliability);
+
+ return reliability;
+}
+
+
+int close_communication_with_fuzzy_scheduler()
+{
+ gw_scheduler_print('I',"Close pipes for communication
   with fuzzy scheduler... OK\n");
+
+ if (outpipe)
+   fclose(outpipe);
+
+ if (inpipe)
+   fclose(inpipe);
+
+ return 0;
+}
+
+
 void gw_scheduler_add_host(gw_scheduler_t * sched,
                            int              hid,
                            int              uslots,
@@ -342,9 +411,7 @@
             mhosts[j].nusage = min_usage / mhosts[j].
               usage;


-        mhosts[j].priority = wfixed * mhosts[j].nfixed +
-                             wrank  * mhosts[j].nrank  +
-                             wusage * mhosts[j].nusage;
```

```
+    mhosts[j].priority = get_host_reliability(sched->hosts
    [uhosts[mhosts[j].uha_id].hid].name,sched->users[sched->
    jobs[jid].ua_id].name);
      }

      qsort(mhosts,nhosts,sizeof(gw_sch_queue_t),queue_cmp)
         ;
```

Listado A.3: Parche para gw_scheduler_hosts.c

Apéndice B

Material presentado a ISKE'09

En este apéndice se muestra el material enviado para la presentación del trabajo de investigación en el congreso ISKE 2009. Su formato original es de póster DIN-A2, pero ha tenido que ser reducido para poder ser adjuntado a la presente memoria.

FUZZY SYSTEM TO EVALUATE RESOURCES RELIABILITY IN A GRID

Germán P. Santos, Luis Garmendia, Rubén S. Montero y Victoria López

Facultad de Informática, Universidad Complutense de Madrid, Madrid, Spain

Introduction

Job scheduling in Computational Grids is a complex task because of the lack of control over the resources. Under these conditions, it is impossible to know exactly which host will perform better for a given job. On the other hand, fuzzy systems have demonstrated in recent years to behave well in this kind of situations, with imprecision, incomplete information or a lack of an exact model of the optimization space.
Here is presented a fuzzy system, built on top of the GridWay meta-scheduler, to evaluate the reliability of each candidate resource. This information is used by GridWay to choose the resource where the job will be dispatched.

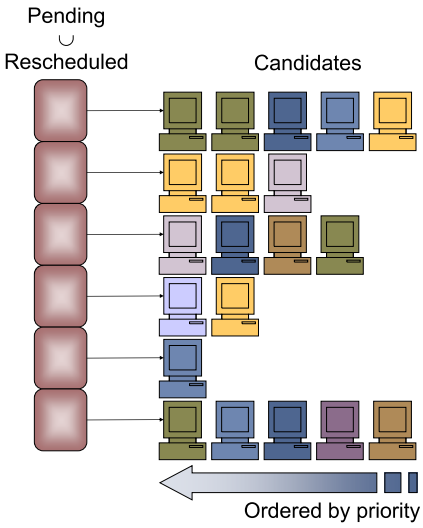
Scheduling policies in GridWay

Grid Scheduling consists on Job Selection Policies and Resource Selection Policies. This paper focuses in the latter. To deal with the dynamism of the Grid, the GridWay meta-scheduler uses *adaptive scheduling,* which consists on allocating pending jobs to grid resources considering the set of available resources, their current status and the accounting information about the already submitted jobs. It can also migrate running jobs to more suitable resources based on events dynamically generated by both the Grid and the application (*adaptive execution*).

The following algorithm describes the process of selecting a host given a job and a user:

```
for each job ∈ Pending ∪ Rescheduled do
    Candidates ← get_candidate_hosts(job,user,resources)
    for each host ∈ Candidates
        priority ← Σ w_i p_i where i ∈ {fixed, usage, rank}
        Ordered ← insert_ordered(priority,host,Ordered)
dispatch job to get_first_element(Ordered)
```

The policies used for the calculation of the priority are **fixed**, which assigns a fixed priority given by the administrator to each Grid resource; **usage**, that considers the execution history of the resource; and **rank**, which calculates the priority of each host using a user supplied expression based on its characteristics as speed or free memory. For example: `RANK = (CPU_MHZ * 2) + FREE_MEM_MB`.



How can Fuzzy Logic improve the scenario

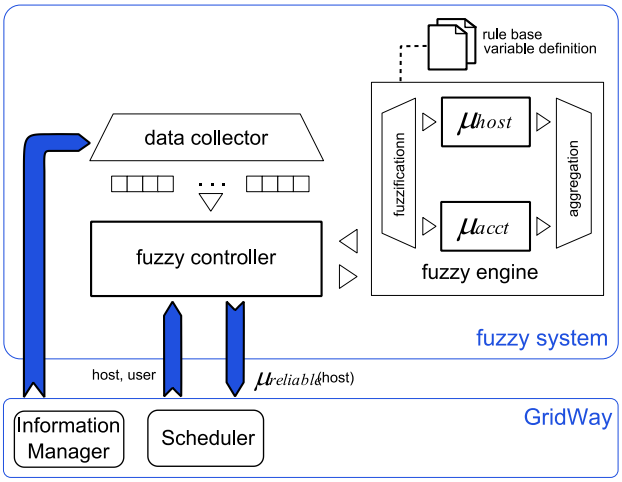
The goal of the fuzzy system is to optimize the priorization of candidate hosts, by defining the reliability of a host *x* as the degree of membership to the fuzzy set reliable:

$$\mu_{reliable}(x) = p\mu_{host}(x) + (1-p)\mu_{acct}(x) \text{ where } p \in [0,1]$$

These two fuzzy sets, μ_{host} and μ_{acct} , are defined by the user. Expressions defining the desired characteristics of the target resource as a combination of restrictions on the model's variables, using the basic operations AND, OR, NOT, are included within the *job template*.
For example, the following configuration could be applied for a CPU intensive job:

```
mon = cpu_load IS low AND cpu_speed IS high
acct = error_rate IS low AND execution_time IS high.
```

Impovements
- User defines preferences in a way that mimics his reasoning abilities
- Better use of accounting information with job-specific rules
- Dealing with different levels of information relevance



Conclusions

In this paper we have presented a new scheme of grid resource selection for the GridWay meta-scheduler based on Fuzzy Logic. This new policy computes the fuzzy set reliable analyzing the input data from monitoring and accounting services. Apart from the advantages of using a soft computing approach, this new method provides some other benefits over the method currently being used. One of them is the possibility of dealing with different levels of information relevance depending on the source. Another one is a better use of accounting information based on an improved resource model and an improved reasoning capability.
There are still things to be addressed. For example, to keep the definition of the linguistic variables meaningful, the fuzzy sets describing its terms should vary depending on the set of available resources. In the future, techniques for adaptation of membership functions will be considered.

This research was supported by Consejería de Educación of Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through BioGridNet Research Program S-0505/TIC/000101; and by Ministerio de Ciencia e Innovación through the research grant TIN2006-02806. It was also partially supported by the Ministerio de Ciencia e Innovación, grant TIN2006-06190 and the Research Group 910149 at Universidad Complutense de Madrid.

Bibliografía

[1] Global grid forum. http://www.ggf.org.

[2] Globus toolkit information services: Monitoring and discovery system. http://www.globus.org/toolkit/mds.

[3] Iec 61131-7 standard: Fuzzy control language specification. Technical report, International Electrotechnical Commission (IEC).

[4] Java compiler compiler (javacc) - the java parser generator. http://javacc.java.net/.

[5] Open grid services infrastructure (ogsi) v1.0. http://forge.gridforum.org/projects/ggf-editor/document/draft-ogsi-service-1/en/1.

[6] Open source fuzzy logic library and fcl language implementation. http://jfuzzylogic.sourceforge.net.

[7] Web services resource framework (wsrf) v1.2. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[8] *The Grid. Blueprint for a New Computing Infrastructure.: Blueprint for a New Computing Infrastructure (Elsevier Series in Grid Computing)*. Morgan Kaufmann, 2. a. edition, December 2003.

[9] Andrea Attanasio, Gianpaolo Ghiani, Lucio Grandinetti, Emanuela Guerriero, and Francesca Guerriero. Operations research methods for resource management and scheduling in a computational grid: a survey. In *High Performance Computing Workshop*, pages 53–81, 2004.

[10] Gleb Beliakov, Ana Pradera, and Tomasa Calvo. *Aggregation Functions: A Guide for Practitioners*. Springer, 2007.

[11] M. Cafaro and G. Aloisio. *Grids, Clouds and Virtualization*. Computer Communications and Networks. Springer, 2010.

[12] Karl Czajkowski, Don Ferguson, Ian Foster, Jeff Frey, Steve Graham, Tom Maguire, David Snelling, and Steve Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution. 2004.

[13] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An introduction to fuzzy control*. Springer, 1996.

[14] Alexander Fölling, Grim Christian, Joachim Lepping, and Alexander Papaspyrou. Job scheduling strategies for parallel processing. chapter Decentralized Grid Scheduling with Evolutionary Fuzzy Systems, pages 16–36. Springer-Verlag, Berlin, Heidelberg, 2009.

[15] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, 2002.

[16] Ian Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.

[17] Jin Huang, Hai Jin, Xia Xie, and Qin Zhang. An approach to grid scheduling optimization based on fuzzy association rule mining. In *E-SCIENCE '05*, pages 189–195. IEEE Computer Society, 2005.

[18] Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. A framework for adaptive execution in grids. *Softw. Pract. Exper.*, 34(7):631–651, 2004.

[19] Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. Gridway framework for adaptive scheduling and execution on grids. *Scalable Computing - Practice and Experience*, 6(3):1–8, 2005.

[20] Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. Evaluating the reliability of computational grids from the end user's point of view. *Journal of Systems Architecture*, (52):727–736, April 2006.

[21] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular norms*. Springer, 2000.

[22] George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, May 1995.

[23] Dalibor Klusáček and Hana Rudová. Alea 2 – job scheduling simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. ICST, 2010.

[24] N. Nedjah and L.M. Mourelle. *Fuzzy systems engineering: theory and practice*. Studies in fuzziness and soft computing. Springer, 2005.

[25] G.P. Santos, L. Garmendia, R.S. Montero, and V. Lopez. Fuzzy system to evaluate resources reliability in a grid environment. In *Proceedings of the 4th International ISKE Conference on Intelligent Systems and Knowledge Engineering*, pages 357–362. World Scientific, November 2009.

[26] Jennifer M. Schopf. Ten actions when grid scheduling. pages 15–23, 2004.

[27] B. Schweizer and A. Sklar. *Probabilistic Metric Spaces*. Elsevier Science, New York, 1983.

[28] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic, and Rajkumar Buyya. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurr. Comput. : Pract. Exper.*, 20(13):1591–1609, 2008.

[29] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybern.*, 18(1):183–190, 1988.

[30] Lofti A. Zadeh. Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, 4(2):103–111, 1996.

[31] Lofti A. Zadeh. Fuzzy sets, fuzzy logic, and fuzzy systems. pages 260–282, 1996.

[32] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[33] Lotfi A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(1):28–44, Jan 1973.

[34] Lotfi A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, parts i, ii, iii. *Information Sciences*, 8,9(3):199–249, 301–357, 43–80, 1975.

[35] Lotfi A. Zadeh. Fuzzy logic, neural networks and soft computing. *Communications of the ACM*, 37(3):77–84, 1994.